UNIVERSIDADE DO ESTADO DE SANTA CATARINA - UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FILIPE RAMOS

ESPECIFICAÇÃO E PROVA DE PROPRIEDADE ACERCA DE AUTÔMATOS FINITOS DETERMINÍSTICOS ASSISTIDAS POR COQ

JOINVILLE - SC

2019

FILIPE RAMOS

ESPECIFICAÇÃO E PROVA DE PROPRIEDADE ACERCA DE AUTÔMATOS FINITOS DETERMINÍSTICOS ASSISTIDAS POR COQ

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Dra. Karina Girardi Roggia
Coorientador: Me. Rafael Castro Gonçalves Silva

JOINVILLE - SC

2019

RESUMO

Os autômatos finitos determinísticos, reconhecedores de linguagens regulares, são muito importantes para a ciência da computação. Na automação industrial, eles podem modelar sistemas orientados a eventos discretos e assíncronos, denominados sistemas a eventos discretos. Nesse contexto, os autômatos são máquinas abstratas que, ao computar uma cadeia de símbolos que representam eventos, descrevem o funcionamento do sistema por meio de estados e transições. Uma das classes de sistemas a eventos discretos é a de sistemas de filas, nos quais há uma fila ou *buffer* e algumas propriedades que devem ser respeitadas no projeto e implementação do sistema. Este trabalho investiga a viabilidade do uso de assistentes de provas a fim de garantir propriedades de sistemas modelados por autômatos finitos determinísticos. Foi adotado o assistente Coq como ferramenta, e serão provadas propriedades do problema produtor-consumidor como estudo de caso.

Palavras-chaves: autômatos finitos determinísticos, assistente de provas, sistemas a eventos discretos, sistemas de filas, produtor-consumidor.

ABSTRACT

Deterministic finite automata, recognizers of regular languages, are deeply relevant in computer science. Regarding to industrial automation, they play a very important role in modeling discrete and asynchronous event-oriented systems, called discrete event systems. In this context, automata are abstract machines that, when computing a string representing a sequence of events, describe the functioning of the system through states and transitions. One of the classes of discrete event systems is formed by queuing systems, where there is a queue or buffer and some properties that must be satisfied by system design and implementation. This paper investigates the feasibility of using proof assistants to assure properties of systems modeled by deterministic finite automata. The Coq assistant has been adopted as a tool, and properties of the producer-consumer problem will be proved as a case study.

Keywords: deterministic finite automata, proof assistant, discrete event systems, queueing systems, producer-consumer.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE SIGLAS E ABREVIATURAS

AFD Autômato finito determinístico

AFND Autômato finito não determinístico

CCI Cálculo de construções indutivas

SED Sistema a eventos discretos

VANT Veículo aéreo não tripulado

SUMÁRIO

1 INTRODUÇÃO

Os autômatos são modelos de máquinas abstratas muito importantes para os estudos da computação teórica. Em particular, os autômatos finitos determinísticos (AFDs) reconhecem as linguagens regulares, aplicadas na ciência da computação e tecnologia da informação (HOPCROFT; MOTWANI; ULLMAN, 2007). Sob outra perspectiva, essa classe de autômatos tem sido utilizada para descrever sistemas do mundo hodierno na automação de indústrias e tecnologias, já que o advento dos sistemas a eventos discretos (SEDs) impôs um novo paradigma, contrário aos sistemas orientados pelo tempo, que são, em geral, muito bem descritos pelos modelos físicos clássicos e modernos (CASSANDRAS; LAFORTUNE, 2008).

Com a finalidade de descrever o funcionamento dos sistemas que são orientados a eventos discretos e assíncronos, eles podem ser modelados por formalismos da matemática discreta, entre os quais se pode citar os autômatos determinísticos, redes de Petri, *timed models* e cadeias de Markov. São exemplos de SEDs que podem ser representados por esses modelos os sistemas de filas, computacionais, de comunicação, manufatura, tráfego, banco de dados e telefonia. Os sistemas de filas, geralmente, envolvem a sincronização de processos manipulando filas ou *buffers*, problema presente na automação industrial. É possível modelar diferentes SEDs com as características dos sistemas de filas a fim de representar plantas industriais com recursos compartilhados entre suas partes. Os AFDs são amplamente usados nesse sentido, já que reduzem o tempo do projeto desses sistemas, permitem formalizá-los e torna mais fácil a manutenção deles (LOPES et al., 2012).

A obtenção de propriedades sobre AFDs específicos pode não ser uma tarefa trivial, e sua formalização, a prova, costuma mostrar-se ainda mais difícil, pois a intuição, muitas vezes, é o único meio utilizado para a garantia das propriedades. Outra técnica utilizada com esse fim é o *model checking*, no qual os instrumentos de verificação – *model checkers* – operam enumerando exaustivamente o espaço dos estados de um modelo. O método é limitante, porque resulta em uma explosão combinatorial, tornando impraticável a verificação de modelos de sistemas reais desse modo (ATHALYE, 2017). Por isso, o emprego de assistentes interativos de provas matemáticas é útil, eles possibilitam a formulação e demonstração das propriedades. Tais assistentes reúnem meios formais e computacionais que propiciam a verificação dos passos da demonstração, auferindo rigor. Entre os assistentes de provas, o Coq destaca-se por ter comunidade extensa, ter uma linguagem funcional de ordem superior e permitir a construção de tipos dependentes. Outras qualidades desse assistente são a possibilidade de programar táticas de provas, desenvolver e carregar módulos

separadamente sem a necessidade de verificá-los novamente e definir notações para melhorar a visualização das definições e provas (BARRAS et al., 1999). O Coq tem se mostrado uma ferramente robusta e bastante aceita pela comunidade acadêmica. O teorema das quatro cores, segundo o qual qualquer mapa pode ser colorido com quatro cores de modo que as regiões adjacentes tenham cores diferentes, foi provado em Coq por Gonthier (2004), resultando em uma prova mais rigorosa em relação às demais demonstrações para o mesmo teorema. Embora o computador tenha sido empregado nas outras provas, essa última destaca-se por dispensar a necessidade de confiar nos programas verificadores de casos particulares e ser expressa na linguagem formal do Coq, à qual se atribuiu maior confiabilidade. Essas e outras características do Coq motivaram a escolha dele para a assistência das provas que este trabalho externa.

Os problemas envolvendo filas em processos concorrentes são muito estudados na ciência da computação e dotados de relevância no que tange aos sistemas de produção industrial, onde a falta de sincronização acarreta eventualmente outros problemas. Se um SED é corretamente modelado por AFDs, deve ser viável identificar nos modelos as eventuais falhas decorrentes da concorrência por recursos distribuídos em filas; senão, apontar os defeitos da representação. Por esse motivo, confere-se valor e importância ao presente estudo.

O objetivo geral deste trabalho de conclusão de curso consiste em provar, mediante o assistente de provas Coq, propriedades sobre sistemas da automação industrial modelados por AFDs. Os objetivos específicos são:

1. introduzir os principais conceitos relativos aos SEDs;
2. apresentar a classe de sistemas de filas;
3. especificar propriedades desejadas para essa classe;
4. empregar o assistente Coq na prova de teoremas referentes a essas propriedades;
5. demonstrar e explanar tais teoremas e exemplificar sua aplicação.

Com a finalidade de atingir os objetivos supracitados, o presente trabalho é estruturados destarte. Primeiramente, o Capítulo 2 introduz o assistente de provas Coq. No Capítulo 3, são introduzidos os SEDs, assim como sua modelagem por AFDs. Já no Capítulo 4, apresentam-se os trabalhos relacionados a este estudo. O capítulo seguinte, 5, apresenta os sistemas de filas, problemas relacionados e as propriedades mais importantes acerca desses SEDs. Em seguida, o Capítulo 6 demonstra como

constatar se sistemas de filas atendem às propriedades. Por fim, são expressas as considerações finais e, depois, as referências bibliográficas.

2 ASSISTENTE DE PROVAS

Um assistente de provas interativo é um artifício de software que auxilia no processo de prova matemática. Diferentemente dos provadores automáticos de teoremas, os assistentes não realizam provas automaticamente com um simples comando do usuário, mas participa com ele no processo oferecendo uma variedade de procedimentos formalizadores. Seu uso também se justifica na necessidade de validar demonstrações, haja vista que detalhes podem passar despercebidos pelo olhar analítico do ser humano. Os assistentes mais robustos implementam a teoria dos tipos, na qual os elementos pertencem a tipos definidos recursivamente, e não a conjuntos.

A teoria dos tipos fornece o encontro da ciência da computação com a matemática e a lógica. Ela é simultaneamente um sistema formal e linguagem de programação que permite raciocinar enquanto se programa. Sua relevância se explica por possuir simplicidade, ser robusta, apresentar a propriedade da decidibilidade, ser uma linguagem funcional, entre outros (LUO, 1994).

Há pelo menos 15 provadores de teoremas no mundo, entre os quais estão os provadores semiautomáticos, ou assistentes de provas. Os assistentes mais utilizados são: HOL, Mizar, PVS, Coq, Isabelle, Agda, Phox, IMPS, Metamath, Lego, NuPRL e Omega-MEGA. Em relação aos sistemas de tipos, o Metamath não é tipado; o HOL, Isabelle, Phox, IMPS e Omega-MEGA permitem apenas tipos não dependentes decidíveis; o PVS e NuPRL, tipos dependentes indecidíveis; os demais, tipos dependentes decidíveis. Apenas o Coq e o NuPRL têm lógica de ordem superior. O Metamath destaca-se por sua lógica quântica, enquanto os demais empregam lógica clássica ou construtivista. Em geral, se um assistente é capaz de verificar a prova de um teorema, seu emprego é válido. A escolha do provador, portanto, baseia-se na subjetividade e no relativo conforto do usuário.

A Seção 2.1 introduz o assistente de provas empregado neste trabalho de conclusão de curso. A escolha do Coq se deve à confiabilidade, utilidade, usabilidade e facilidade desse assistente.

2.1 COQ

Coq é um dos assistentes de provas mais utilizados atualmente, permitindo a construção de tipos dependentes e polimórficos, lógica de ordem superior, etc. A linguagem formal deste assistente de provas é o cálculo de construções indutivas (CCI), no qual há duas variedades de objetos, os tipos e os termos. Os tipos são classes a que pertencem os termos – o número 1 é um termo do tipo natural – e

podem ser termos de outros tipos. Nessa linguagem formal, a notação para um termo x do tipo X é $x : X$, com a qual se deve habituar para utilizar o Coq (BARRAS et al., 1999).

No Coq os tipos são definidos indutivamente pelos comandos `Inductive`, `Fixpoint` e `Record`. O primeiro permite definir termos destarte:

$$\texttt{Inductive } ident \texttt{ : } term \texttt{ :=}$$
$$ident_1 \texttt{ : } term_1 \texttt{ | } ident_2 \texttt{ : } term_2 \texttt{ | ... | } ident_n \texttt{ : } term_n\texttt{.}$$

em que $ident$ é o nome do objeto sendo definido e $term$ é seu tipo. Cada $ident_i$ e $term_i$ são os respectivos nome e tipo do i-ésimo construtor ($\forall i = 1..n$). O comando `Fixpoint` define funções de forma recursiva baseada nos m argumentos:

$$\texttt{Fixpoint } ident \texttt{ (} ident_1 \texttt{ : } term_1\texttt{) (} ident_1 \texttt{ : } term_1\texttt{) ... (} ident_m \texttt{ : } term_m\texttt{)}$$
$$\texttt{ : } term \texttt{ := } term'\texttt{.}$$

A definição usando esse comando deve garantir que a recursão sempre para. Para tanto, o usuário precisa especificar a recursão mediante a estrutura `match with`:

$$\begin{aligned} term' := \texttt{match } & ident_1\texttt{, } ident_2\texttt{, ..., } ident_m \texttt{ with} \\ & a_1^1\texttt{, } a_2^1\texttt{, ..., } a_m^1 \texttt{ => } term_1' \texttt{ |} \\ & a_1^2\texttt{, } a_2^2\texttt{, ..., } a_m^2 \texttt{ => } term_2' \texttt{ |} \\ & \vdots \\ & a_1^k\texttt{, } a_2^k\texttt{, ..., } a_m^k \texttt{ => } term_k' \\ & \texttt{end} \end{aligned}$$

gerando uma análise de caso para o valor de cada $ident_i$. Assim, se $\langle ident_1, ident_2, ..., ident_m\rangle$ é $\langle a_1^j, a_2^j, ..., a_m^j\rangle$, o retorno da função é $term_j'$ ($\forall i = 1..m, j = 1..k$). Quando não é necessário haver recursão, pode-se substituir o comando pelo `Definition`. Já o comando `Record` aproxima o usuário das linguagens de programação comuns, ao definir estruturas parecidas com as quais se desenvolve nelas:

$$\texttt{Record } ident\ params \texttt{ : } term \texttt{ := } ident_0 \texttt{ \{}$$
$$ident_1 \texttt{ : } term_1\texttt{;}$$
$$ident_2 \texttt{ : } term_2\texttt{;}$$
$$\vdots$$
$$ident_n \texttt{ : } term_n \texttt{ \}.}$$

em que $params$, $ident_0$, $ident_i$ e $term_i$ são, respectivamente, os parâmetros ou argumentos, nome do construtor (opcional), nome e tipo do i-ésimo campo ($\forall i = 1..n$),

havendo n campos. Na linguagem do Coq, o i-ésimo campo de um *record* é obtido com esta sintaxe: $ident_i\ x$, em que $x : ident$. Há outras maneiras de definir termos a partir desses comandos, mas as apresentadas são suficientes para os objetivos do presente trabalho.

A sintaxe exposta é a combinação do CCI com a linguagem de especificação Gallina, bastante intuitivos. Para provar teoremas e lemas no Coq, além de definir os tipos que serão usados nas proposições, é necessário utilizar o comando `Theorem` ou `Lemma`:

$$\texttt{Theorem}\ ident\ \texttt{:=}\ ident'\texttt{.}\ \texttt{Proof.}\ tactic_1\texttt{.}\ tactic_2\texttt{.}\ \ldots\texttt{.}\ tactic_m\texttt{.}\ \texttt{Qed.}$$

com m táticas. As táticas são o cerne da interatividade do Coq e facilitam o desenvolvimento de provas. Elas implementam um raciocínio que parte da conclusão para as premissas, denominadas objetivo e subobjetivos respectivamente. Uma tática, aplicada corretamente sobre o objeto, gera subobjetivos que o substituem (BARRAS et al., 1999). O Quadro 1 apresenta as principais táticas do Coq utilizadas por este trabalho.

Quadro 1 – Principais táticas do Coq

Nome	Descrição
`intros`	Introduz variáveis
`simpl`	Simplifica expressão
`reflexivity`	Valida expressões $a = a$
`rewrite`	Reescreve relações de igualdade
`apply`	Aplica um lema, teorema ou axioma
`symmetry`	Troca $a = b$ por $b = a$
`injection`	Infere $a = b$ a partir de $f(a) = f(b)$, sendo f injetora
`omega`	Completa prova a partir de igualdades e desigualdades envolvendo inteiros
`induction`	Faz indução sobre uma variável

Fonte: Elaborado pelo autor com base em Barras et al. (1999), 2019.

Outras táticas importantes são: `unfold` – para expandir expressões reduzidas –, `discriminate` – usada quando existe uma igualdade falsa no contexto, como $1 = 0$ – e `inversion`. Esta última faz o Coq analisar outras condições para que uma hipótese seja verdadeira. A fim de exemplificar o uso das táticas do Coq, é provado o teorema

```
Theorem plus_n_O : ∀ n : nat, n = n + 0.
```

em que nat é o tipo dos números naturais. Começa-se removendo o quantificador universal e introduzindo a variável `n`:

```
Proof. intros n.
```

Agora, inicia-se a indução sobre `n`:

```
induction n as [|n' IHn'].
```

em que os símbolos após o "`|`" nomeiam as variáveis a serem utilizadas no passo indutivo. Para a base `n = 0`, basta simplificar e aplicar a tática `reflexivity`. Como esta já simplifica a expressão, não é necessário aplicar `simpl`. No passo indutivo, tem-se `n = S n'`, sendo `S x` a função sucessor de `x : nat`. O objetivo atual é

```
S n' = S n' + 0
```

Simplificando com `simpl`, obtém-se

```
S n' = S (n' + 0)
```

Agora se pode aplicar a hipótese de indução `IHn' = (n' = n' + 0)` mediante a tática

```
rewrite <- IHn'.
```

com `<-` indicando que o lado direito da hipótese `IHn'` deve ser reescrito pelo esquerdo no objetivo, o que gera

```
S n' = S n'
```

Por fim,

```
reflexivity. Qed.
```

como se queria demonstrar.

O capítulo seguinte, 3, conceitua os sistemas a eventos discretos e autômatos finitos determinísticos, objetos das demonstrações do Capítulo 6. Essas provas foram obtidas e validadas no ambiente do assistente Coq.

3 SISTEMAS A EVENTOS DISCRETOS

Segundo Lafortune (2019, p. 142, tradução do autor)

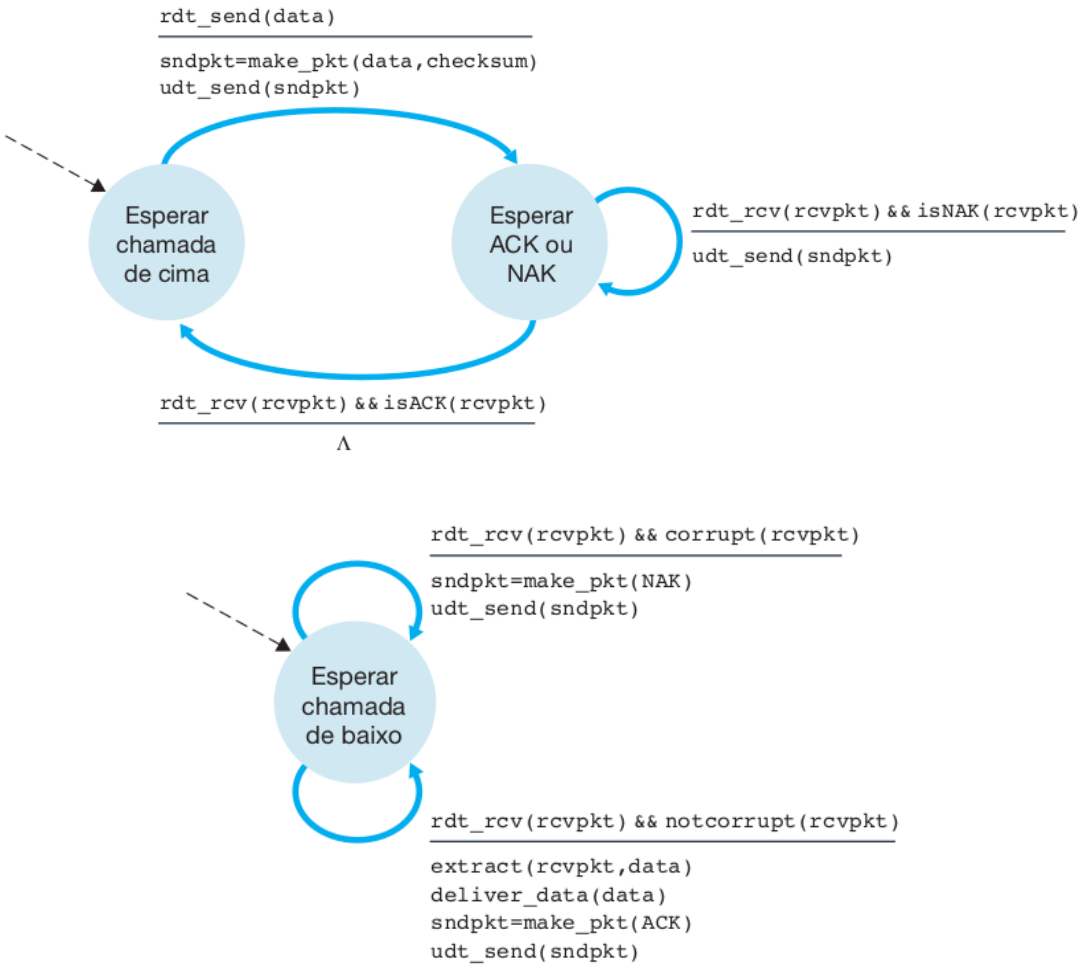
> Sistemas a eventos discretos (SEDs) são sistemas dinâmicos com duas características definidoras: seus espaços de estados são discretos e potencialmente infinitos, e sua dinâmica é orientada a eventos, em vez de tempo. (...) Eventos que ocorrem de forma assíncrona (em geral) causam um salto no espaço de estados, de um estado para outro.

Cassandras e Lafortune (2008) argumentam que os SEDs, diferentemente dos sistemas orientados pelo tempo e governados pelas leis da natureza, não podem ser modelados e estudados usando equações diferenciais. Os SEDs requerem outras técnicas de análise, ferramentas de projeto, métodos de teste, entre outros.

O advento das tecnologias de comunicação, sensoriamento e computação impulsionou a caracterização desta nova classe de sistemas. Os SEDs, por não serem adequadamente modelados pelas equações das leis da física clássica e moderna, demandam novos paradigmas de modelagem. “Intuitivamente, nós podemos pensar em um modelo como um dispositivo que simplesmente duplica o comportamento do próprio sistema” (CASSANDRAS; LAFORTUNE, 2008, p. 3, tradução do autor). Um modelo descreve um sistema sob uma ótica quantitativa, que é, muitas vezes, mais adequada que uma simples e subjetiva descrição qualitativa. A construção de um modelo considera um conjunto de variáveis de entrada e saída, relacionadas por intermédio de uma função. Assim sendo, os modelos proporcionam a habilidade predizer os comportamentos de sistemas, o que os confere importância. Formas de modelar sistemas que evoluem com base em eventos assíncronos e discretos são discutidas neste capítulo.

Cassandras e Lafortune (2008) citam, como exemplos de SEDs, os sistemas de filas, computacionais, de comunicação, de manufatura e de tráfego. A exemplo disso, protocolos de comunicação em redes de computadores são frequentemente modelados por SEDs, como indica a Figura 1. O presente trabalho enfoca os sistemas relacionados à automação industrial, para os quais a modelagem de filas é conveniente. A espera de entidades por um recurso em uma fábrica é exemplo que justifica isso, porque a abstração por meio de filas ou *buffers* é muito útil nesse caso. O Capítulo 5 explanará sobre essa e outras situações em que se pode abstrair a noção de filas.

Figura 1 – Protocolo para um canal de redes de computadores com erros de bits: lado remetente (acima) e destinatário (abaixo)



Fonte: KUROSE, J.; ROSS K. **Redes de computadores e a internet**: uma abordagem top-down. 6. ed. São Paulo: Pearson, 2013.

Na Figura 1, tem-se representado um protocolo de redes em que há estados de espera e transições que dependem de eventos discretos – neste caso, procedimentos computacionais. A representação supracitada é uma evidência do caráter de SED do protocolo, e a figura é um diagrama de estados, que será introduzido mais adiante.

SEDs são comumente modelados por redes de Petri e autômatos determinísticos. Redes de Petri são uma classe de grafos direcionados que têm duas variedades de nós: posições e transições. Nelas, os arcos podem ligar dois nós apenas se forem diferentes – não pode haver arcos entre duas posições, por exemplo – e têm pesos que indicam a quantidade de tokens que serão consumidos ou produzidos. Geralmente, os tokens, posições e transições representam recursos, condições e eventos respectivamente e permitem modelar atividades paralelas, protocolos de comunicação, sistemas de produtor-consumidor com prioridade, etc. (MURATA, 1989) Este trabalho de conclusão de concurso, todavia, tem enfoque na representação por autômatos finitos

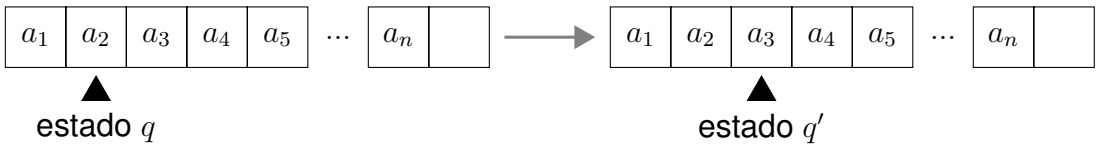
determinísticos. A Seção 3.1 introduz os principais conceitos desses modelos.

3.1 AUTÔMATOS FINITOS DETERMINÍSTICOS

Autômato – palavra derivada do termo em latim *automatu* – é “maquinismo que se move por meios mecânicos” e “imita os movimentos humanos” (FERREIRA, 2010, p. 81). Esse termo tem sido usado na ciência da computação desde a década de 1930 para descrever importantes modelos estudados posteriormente pela teoria dos autômatos, que aborda máquinas de Turing, por exemplo. Nesse contexto, um autômato é uma máquina abstrata descrita matematicamente e idealizada em termos de limitações físicas (HOPCROFT; MOTWANI; ULLMAN, 2007).

Um autômato finito determinístico (AFD), ou máquina de estados finitos determinística, é uma máquina dotada de fita, unidade de controle e função de transição. A fita de um autômato é um espaço ilimitado utilizado para armazenar uma sequência de símbolos que serão lidos e computados. Já a unidade de controle contém as variáveis do estado atual da máquina, que servem de parâmetro para a computação da função de transição. Para determinar o estado de um autômato, há uma cabeça de leitura sobre a fita e um conjunto de elementos abstratos que norteiam a evolução do funcionamento da máquina: os estados (HOPCROFT; MOTWANI; ULLMAN, 2007; MENEZES, 2005). A Figura 2 esquematiza a transição de estado de um AFD com a cabeça de leitura inicialmente posicionada sobre a segunda célula da fita.

Figura 2 – Transição de um AFD



Fonte: Elaborada pelo autor, 2019.

Durante a computação de uma cadeia de entrada, o AFD lê o símbolo da célula atual da fita, apontada pela cabeça de leitura, e avança o cabeçote em uma posição para a direita. Inicialmente, ao receber uma entrada, a cabeça de leitura estará posicionada na extremidade esquerda da fita e, por conseguinte, da cadeia de símbolos. Se a computação de um símbolo lido não acarretar um estado definido ou a cabeça de leitura estiver posicionada sobre uma célula vazia, o funcionamento do autômato será interrompido.

Na ciência da computação, os AFDs formam a base de alguns componentes de software e partes de compiladores. Um artifício muito empregado no desenvolvimento de software são as expressões regulares, que podem ser convertidas em AFDs

e permitem encontrar padrões em textos (HOPCROFT; MOTWANI; ULLMAN, 2007). Já no contexto dos SEDs, os AFDs são modelos matemáticos que descrevem sistemas com base nos eventos que podem ocorrer. Dessa maneira, as cadeias de símbolos que são enviadas à entrada dos AFDs constituem sequências de eventos cuja computação resulta em uma descrição do sistema baseada nas variáveis de controle da máquina de estados (CASSANDRAS; LAFORTUNE, 2008).

3.1.1 Definição formal

A definição de AFD que segue foi inspirada e adaptada de Hopcroft, Motwani e Ullman (2007) e Cassandras e Lafortune (2008).

Um AFD G é uma quíntupla

$$\langle Q, E, \delta, q_0, Q_m \rangle \tag{3.1}$$

em que

Q é o conjunto finito de estados

E é o conjunto finito de símbolos ou eventos

$\delta : Q \times E \nrightarrow Q$ é a função de transição

q_0 é o estado inicial

$Q_m \subseteq Q$ é o conjunto de estados marcados

A função de eventos ativos de G, que será denotada por $\Gamma_G : Q \to 2^E$, relaciona cada estado com os eventos possíveis a partir dele. Formalmente, para todo estado $q \in Q$

$$\forall e \in E, e \in \Gamma_G(q) \Leftrightarrow \delta(q, e) \in Q$$

isto é, $e \in \Gamma_G(q)$ se e somente se $\delta(q, e)$ é definido.

Ao fecho de Kleene sobre um conjunto de eventos E, denotado por $E^\star$, pertencem todas as possíveis cadeias de eventos pertencentes a E. Uma cadeia de eventos é uma sequência finita $e_1 e_2 ... e_{|w|}$ em que $e_i \in E$ ($\forall i = 1..|w|$). Quando $|w| = 0$, a cadeia é dita vazia e será simbolizada por ε.

A função de transição estendida $\hat{\delta} : Q \times E^\star \nrightarrow Q$ é definida recursivamente destarte:

$$\hat{\delta}(q, w) = \begin{cases} q & \text{se } w = \varepsilon \\ \hat{\delta}(\delta(q, e), w') & \text{se } w = ew' \end{cases}$$

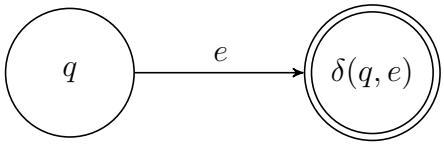
e terá valor indefinido quando $\delta(q, e)$ assim for.

Embora a formulação matemática de um AFD seja muito útil e importante para formalizar SEDs e provar propriedades destes, a visualização do funcionamento destes autômatos é facilitada com a representação por diagramas de estados. Ademais, tendo em vista que o presente trabalho dispõe de diversos destes diagramas, a subseção 3.1.2 introduz essas representações.

3.1.2 Diagrama de estados

Para auxiliar na visualização das transições entre os estados dos autômatos, os AFDs são comumente representados por diagramas de estados. Nessa representação, os estados são nós de uma estrutura semelhante a de grafos, e as transições, arestas que interligam dois nós, conforme a Figura 3. Representam-se as transições cuja origem e destino são o mesmo estado por *loops*: arestas que partem de um nó e terminam no mesmo.

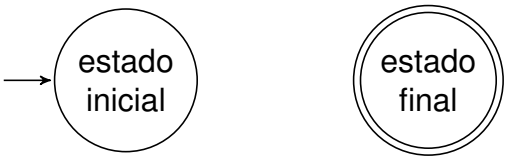
Figura 3 – Representação da transição de estados em um diagrama



Fonte: Elaborada pelo autor, 2019.

Nesta classe de diagramas, os estados inicial e final podem ser destacados de alguma forma. Para este trabalho, uma seta sem nó de origem aponta sempre para o nó do estado inicial, e uma circunferência dupla enfatiza o de um estado final, como demonstra a Figura 4.

Figura 4 – Representação de estados inicial (à esquerda) e final (à direita) em um diagrama



Fonte: Elaborada pelo autor, 2019.

Pode-se citar outros aspectos desta representação de autômatos: a possibilidade de adicionar rótulos aos nós, a opção de omitir os nomes dos estados nos nós quando não forem necessários e a aglutinação de transições que partem e terminam no mesmo estado em uma mesma aresta, com os símbolos separados por vírgula.

3.1.3 Linguagem marcada

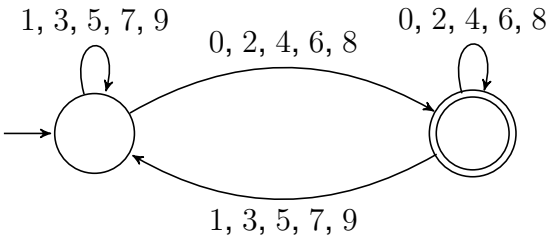
A linguagem marcada por um AFD G é o conjunto

$$L_m(G) = \{w \in E^\star \mid \hat{\delta}(q_0, w) \in Q_m\}$$

Sendo assim, a linguagem marcada pelo autômato são todas as cadeias de eventos que o fazem transicionar do estado inicial a um estado marcado. Isso significa que, quando uma cadeia $w \in L_m(G)$ for posicionada na fita do autômato, a computação de cada evento, da esquerda para a direita da sequência, sempre resultará em um estado definido e terminará em um estado marcado.

Denomina-se linguagem regular a linguagem marcada por qualquer AFD (HOPCROFT; MOTWANI; ULLMAN, 2007). Desse modo, este trabalho de conclusão de curso versará sobre linguagens exclusivamente regulares, a exemplo das quais é possível citar o conjunto dos números naturais pares. Sabendo que um número par é aquele cujo último algarismo – da esquerda para a direita – pertence a $\{0, 2, 4, 6, 8\}$, pode-se construir o AFD da Figura 5, demonstrando a validade da afirmação.

Figura 5 – Diagrama de estados para um AFD que reconhece os números naturais pares



Fonte: Elaborada pelo autor, 2019.

Quando um autômato se destina a verificar padrões em cadeias de eventos, ou palavras, diz-se que ele é um formalismo reconhecedor (MENEZES, 2005) e, portanto, o autômato da Figura 5 reconhece todos os números naturais pares. No que tange aos SEDs, alcançar um estado marcado implica a conclusão de alguma tarefa (CASSANDRAS; LAFORTUNE, 2008). A impossibilidade de concluir tarefas, os bloqueios ocorrem quando não é possível sair de um estado não marcado e chegar a um marcado, tema da subseção 3.1.4.

3.1.4 Linguagem gerada e bloqueios

No contextos dos SEDs, designa-se linguagem gerada por um AFD G o conjunto

$$L(G) = \{w \in E^\star \mid \hat{\delta}(q_0, w) \in Q\}$$

Neste caso, a linguagem são todas as cadeias de eventos que fazem o autômato transicionar do estado inicial a um estado definido.

Caso o autômato G chegue a um estado $q \notin Q_m$ tal que $\Gamma_G(q) = \varnothing$, diz-se que há um *deadlock*, já que não existe evento que pode ser executado a partir de q. Um *livelock* se configura quando, mesmo não havendo *deadlock*, não é possível alcançar um estado marcado. Se qualquer bloqueio acontece

$$\overline{L_m(G)} \subset L(G)$$

em que $\overline{L_m(G)}$ é o conjunto de todos os prefixos de todas as cadeias pertencentes a $L_m(G)$, ou

$$\overline{L_m(G)} = \{w \in E^\star \mid \exists w' \in E^\star, ww' \in L_m(G)\}$$

Isso é devido à constatação de que, tendo sido executada uma cadeia de eventos $w \in E^\star$ e estando em um bloqueio, w pertencerá a $L(G)$, mas não será prefixo de alguma cadeia da linguagem marcada: não haverá $w' \in E^\star$ tal que $ww' \in L_m(G)$, por definição (CASSANDRAS; LAFORTUNE, 2008).

3.1.5 Autômatos finitos não determinísticos

Um autômato finito não determinístico (AFND) é um modelo diferenciado com a capacidade de estar em mais de um estado ao mesmo tempo, segundo Hopcroft, Motwani e Ullman (2007). Essa característica configura a predição dessas máquinas abstratas, permitindo estimar os possíveis estados em que um SED se encontra. Para um AFND $G = \langle Q, E, \delta, q_0, Q_m \rangle$, a função de transição é do tipo $\delta : Q \times Q \to 2^Q$, retornando um conjunto de estados.

A função de transição estendida $\hat{\delta} : Q \times E^\star \to 2^Q$ é tal que para todo estado $q \in Q$, cadeia $w \in E^\star$ e evento $e \in E$

$$\hat{\delta}(q, w) = \begin{cases} \{q\} & \text{se } w = \varepsilon \\ \bigcup\limits_{q' \in \delta(q,e)} \hat{\delta}(q', w') & \text{se } w = ew' \end{cases}$$

Com ela, as linguagens gerada e marcada por um AFND G são respectivamente

$$L(G) = \{w \in E^\star \mid \hat{\delta}(q_0, w) \neq \varnothing\}$$

e

$$L_m(G) = \{w \in E^\star \mid \hat{\delta}(q_0, w) \cap Q_m \neq \varnothing\}$$

isto é, os conjuntos de todas as cadeias de eventos que fazem o autômato transicionar a algum estado qualquer e marcado respectivamente.

A vantagem de empregar AFNDs reside no fato de que a modelagem de SEDs é facilitada. Quando a execução de um evento resulta em diferentes significados, pode-se designar a transição levando a um conjunto de estados cujo nome expressa algum resultado. Por exemplo, supondo que um sistema contém três produtores e um sensor que indica a produção de algum item, um jeito de modelar o sistema é por meio de uma máquina abstrata que vai do estado $\langle n_1, n_2, n_3 \rangle$, ao ser executado um evento de produção, para os pertencentes a $\{\langle n_1 + 1, n_2, n_3 \rangle, \langle n_1, n_2 + 1, n_3 \rangle, \langle n_1, n_2, n_3 + 1 \rangle\}$, em que n_1, n_2 e n_3 são os respectivos números de itens produzidos pelo primeiro, segundo e terceiro agentes.

Hopcroft, Motwani e Ullman (2007) demonstram a conversão de qualquer AFND G para um AFD G' usando a construção de subconjuntos, de forma que $L(G) = L(G')$ e $L_m(G) = L_m(G')$. Assim, convertendo os modelos, é possível usar os teoremas do Capítulo 6 para verificar propriedades de sistemas modelados por AFND.

3.1.6 Função de transição total

Haja vista que não é possível formular funções parciais no assistente de provas Coq, algumas mudanças na definição de AFD supracitada são necessárias a fim de representar AFDs nessa ferramenta. A começar, é inevitável alterar a função de transição de um AFD G para torná-la total. Seja $\delta' : Q \cup \{\otimes\} \times E \to Q \cup \{\otimes\}$ a seguinte função total:

$$\delta'(q, e) = \begin{cases} \delta(q, e) & \text{se } \delta(q, e) \in Q \\ \otimes & \text{senão} \end{cases}$$

em que $\otimes$ é um estado novo, não pertencente a Q: o estado de ralo.

O autômato G é muito semelhante ao

$$G' = \langle Q \cup \{\otimes\}, E, \delta', q_0, Q_m \rangle$$

uma vez que a única diferença entre eles é que, em G', ao realizar-se uma transição que seria indefinida em G, alcança-se um estado do qual não se pode sair.

Como a função δ' é total, tem-se que

$$L(G') = E^\star$$

em termos do que se estabeleceu como linguagem gerada anteriormente. Pode-se, não obstante, modificar a definição dela de forma que G e G' sejam equivalentes em se tratando de linguagens:

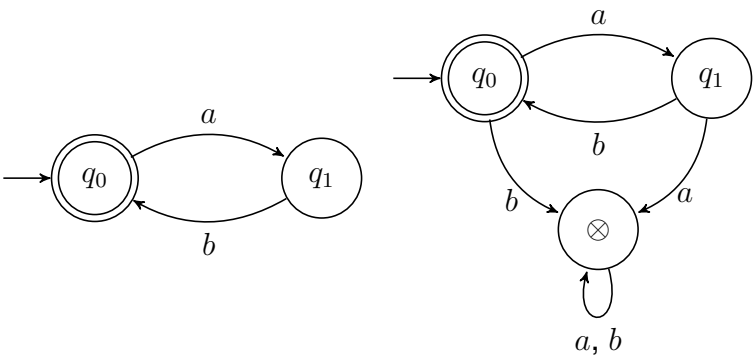
$$L'(G') = \{w \in E^\star \mid \hat{\delta}(q_0, w) \in Q \wedge \hat{\delta}(q_0, w) \neq \otimes\} \tag{3.2}$$

é a linguagem gerada pelo autômato G'. Então

$$L'(G') = L'(G) = L(G)$$

É possível visualizar as formulações de um AFD usando função de transição parcial e total na Figura 6, que a exemplifica para um AFD de alfabeto $\{a, b\}$.

Figura 6 – Um AFD simples (à esquerda) e seu correspondente com função de transição total (à direita)



Fonte: Elaborada pelo autor, 2019.

Isso posto, é evidente que um AFD cuja função de transição não é total pode ser modelado matematicamente utilizando funções totais. Esse fato é importante para que se possa representá-lo na linguagem do Coq.

3.1.7 Formulação em Coq

Todas as definições acerca de AFDs supracitadas utilizam conjuntos, mas, para representar os autômatos no assistente de provas Coq, é necessário que sejam usados tipos. Assim, estados e eventos terão tipos específicos.

Conforme a Subseção 3.1.5, a formulação de AFDs em Coq precisa considerar o estado de ralo para que a função de transição seja sempre total. Nesse sentido, pode-se definir os estados como do seguinte tipo paramétrico:

```
Inductive state {Q : Type} := sink_state | proper_state (q : Q).
```

sendo *proper state* – estado próprio – qualquer estado do autômato exceto o de ralo. Ao invés de usar apenas o tipo `Q`, essa definição é importante para facilitar a caracterização das linguagens geradas, já que o estado de ralo tem de estar bem delimitado para defini-las, como demonstra a Equação 3.2.

A função de transição será do tipo `Q→E→state`, e não `state→E→state`, o que permitiria quaisquer transições partindo do estado de ralo. Ademais, é importante especificar quais estados são marcados com uma função do tipo `Q→bool` que retorna `true` se e somente se o estado for marcado.

Com isso, um autômato determinístico `g : dfa` terá estes campos:

- `Q`: o tipo dos estados próprios
- `E`: o tipo dos símbolos ou eventos
- `transition : Q→E→@state Q`: a função de transição
- `initial_state : Q`: o estado inicial
- `is_marked : Q→bool`: a função de demarcação dos estados

Esta definição permite que termos do tipo `dfa` sejam, com efeito, autômatos infinitos determinísticos. O presente trabalho, porém, assume que os tipos dos estados e eventos são sempre finitos. A implementação do tipo `dfa` proposta é apresentada pela Figura 7.

Figura 7 – Implementação do tipo dos AFDs em Coq

```
Record dfa (Q E : Type) := {
  transition: Q -> E -> @state Q;
  initial_state: Q;
  is_marked: Q -> bool
}.
```

Fonte: Elaborada pelo autor, 2019.

A definição de AFD anterior é equivalente a esta. Sejam q_1, q_2, ... e q_n, q^* e e_1, e_2, ... e e_m, respectivamente, todos os estados, o estado inicial e todos os possíveis símbolos de um AFD, faz-se a construção dos tipos que seguem:

$$\texttt{Inductive Q := } q_1 \texttt{ | } q_2 \texttt{ | ... | } q_n \texttt{.}$$
$$\texttt{Inductive E := } e_1 \texttt{ | } e_2 \texttt{ | ... | } e_m \texttt{.}$$

que são os respectivos tipos dos estados e eventos do autômato na definição para o Coq. Seja $\delta : \{q_1, q_2, ..., q_n\} \times \{e_1, e_2, ..., e_m\} \nrightarrow \{q_1, q_2, ..., q_n\}$ a função de transição de estados tal que

$$\delta(q, e) = \begin{cases} q' & \text{se } \texttt{transition } q \ e = \texttt{proper\_state } q' \\ \text{indefinido} & \text{senão} \end{cases}$$

ou, por outro lado, seja `transition : Q→E→@state Q` assim definida na linguagem do Coq:

```
Definition transition (q : Q) (e : E) : state :=
match q, e with
q', e' => proper_state δ(q', e') |
_, _ => sink_state
end.
```

sendo $x = (q', e')$ todo par tal que $\delta(x)$ é definido. Além disso, determina-se o conjunto de estados marcados:

$$Q_m = \{q_i \mid i \in \{1, 2, ..., n\} \wedge \texttt{is\_marked}\ q_i\ =\ \texttt{true}\}$$

ou, de outra parte, constrói-se a função `is_marked`:

```
Definition is_marked (q : Q) : bool :=
match q with
q_m => true |
_ => false
end.
```

em que q_m é todo estado pertencente a Q_m. Portanto, o autômato $\langle\{q_1, q_2, ..., q_n\}, \{e_1, e_2, ..., e_m\}, \delta, q^*, Q_m\rangle$ pode ser formulado em termos dos elementos `Q`, `E`, `transition` e `is_marked` e vice-versa.

Toda sequência de eventos será representada por uma lista, e a função de transição estendida `extended_transition` será do tipo `dfa→@state Q→list E→@state Q` e definida recursivamente. Se o estado de origem for o de ralo, ele será o próprio retorno da função; senão, `extended_transition` computará a função de transição para a cabeça da lista de eventos e fará recursão destarte:

```
extended_transition g (proper_state q) [] := proper_state q;
extended_transition g (proper_state q) e::w :=
extended_transition g (transition q e) w
```

Por definição, uma lista de eventos `w` será gerada por `g`, ou `g ==> w`, se

`extended_transition g (proper_state initial_state) w` $\neq$ `sink_state`

ou seja, se a computação da função de transição estendida da lista partindo do estado inicial não resultar no estado de ralo, em conformidade com a Equação 3.2. Sendo G o AFD definido conforme a Equação 3.1 e equivalente ao `g`, nota-se

$$L(G) = \{a_1a_2...a_n \in E^\star \mid \texttt{g ==> [}a_1\texttt{;}a_2\texttt{;...;}a_n\texttt{]}\}$$

evidenciando também a correlação entre as diferentes definições supracitadas para a mesma classe de máquinas abstratas.

Com intenção de exemplificar o uso dessa implementação, sejam os tipos

```
Inductive states : Type := q0 | q1 | q2.
Inductive events : Type := a | b.
```

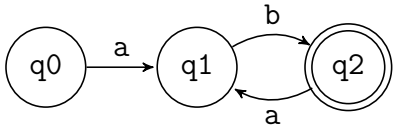
e as funções

```
Definition transition' (q:states) (e:events) : state :=
match q, e with
q0, a => proper_state q1 |
q1, b => proper_state q2 |
q2, a => proper_state q1 |
_, _ => sink_state
end.
Definition is_marked' (q:states) : bool :=
match q with
q2 => true |
_ => false
end.
```

Com isso, o AFD da figura 8 é definido em Coq deste modo:

```
Definition dfa := {|
transition := transition';
initial_state := q0;
is_marked := is_marked'
|}.
```

Figura 8 – AFD cujos tipos dos estados e eventos são `states` e `events` respectivamente



Fonte: Elaborada pelo autor, 2019.

O diagrama da Figura 8 não ilustra o estado de ralo, `sink_state`, evidenciando apenas os estados próprios, `proper_state`.

4 TRABALHOS RELACIONADOS

Doczkal, Kaiser e Smolka (2013) apresentam uma teoria construtiva de linguagens regulares em Coq na qual provam o teorema de Kleene, o lema do bombeamento, a unicidade do autômato determinístico mínimo, o teorema de Myhill-Nerode e propriedades sobre expressões regulares. No trabalho, os autores definem as expressões regulares de maneira indutiva no Coq e formalizam autômatos finitos determinísticos (AFDs) e não determinísticos (AFNDS) como dois tipos `record`. Para garantir que os tipos dos estados e símbolos dos autômatos sejam finitos, eles utilizam uma extensão do Coq e o tipo `char` respectivamente. O trabalho resultou em cerca de 1400 linhas de código e 170 lemas.

Braibant e Pous (2011) propõem uma tática para resolver equações e inequações em álgebras de Kleene, uma porção de relações binárias decidíveis não triviais constituída pelas constantes e operadores das linguagens regulares. Para verificar se duas expressões regulares expressam a mesma linguagem, os autores desenvolveram um algoritmo que constrói um AFND com transições ε para cada expressão, converte-os para AFDs e verifica a equivalência entre eles. O desenvolvimento foi feito no Coq, bem como a prova de que ele é correto. O trabalho originou um algoritmo de 10000 linhas eficiente, se comparado seu desempenho de tempo com o dos outros algoritmos.

Outro trabalho relacionado que implementa autômatos usando tipos `record` do Coq é o de Athalye (2017), que versa sobre autômatos de entrada e saída: *IO automata*. Tais autômatos são, como o autor define, modelos de componentes de sistemas distribuídos, servindo de formalismos matemáticos para garantir o corretismo de projetos e implementações de algoritmos distribuídos e protocolos. O trabalho assemelha-se a este no sentido de modelar possíveis sistemas a eventos discretos (SEDs) usando autômatos e de provar características sobre autômatos no assistente Coq. O autor destaca a explosão combinatorial decorrente do *model checking*, o que faz essa técnica ser inutilizável na verificação de sistemas reais.

Sistemas híbridos são sistemas orientados a eventos e tempo. Nesse contexto, Tveretina (2009) sugere decompor o espaço dos estados em uma grade retangular, com o objetivo de reduzir custos na verificação de segurança. A tarefa consiste em buscar resposta à pergunta: “há como chegar a um estado não seguro?”. Para tanto, a autora implementou um modelo de autômato híbrido em Coq e utilizou-o para modelar um controlador de portão de passagem de trem. Haja vista a semelhança entre SEDs e sistemas híbridos, esse trabalho relacionado evidencia a relevância do

Coq na verificação de segurança para sistemas variados.

Os trabalhos supracitados compõem uma revisão de literatura acerca da modelagem e verificação de SEDs mediante o assistente de provas Coq. Tendo em vista a relação entre AFDs e SEDs, entre modelos de autômatos e entre linguagens regulares e AFDs, os quatro trabalhos relacionados correlacionam-se com o presente trabalho de conclusão de curso. Não se detectou publicação a respeito da especificação e prova de propriedades de sistemas de filas, o que garante ao presente trabalho relativa originalidade.

5 PROBLEMAS DE FILAS

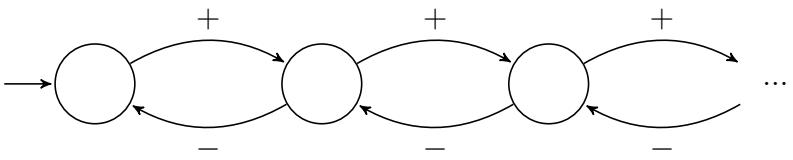
Os sistemas de filas constituem importante classe de sistemas a eventos discretos (SEDs) em que entidades ou objetos devem esperar para obter um determinado recurso. Há três elementos básicos que compõem esses sistemas:

- consumidor: entidade que espera por um recurso;
- servidor: o recurso que é aguardado;
- fila: o espaço em que os consumidores esperam.

Os recursos são genericamente denominados servidores por geralmente proverem serviço (CASSANDRAS; LAFORTUNE, 2008). O presente trabalho aborda uma ótica abstrata dos sistemas de filas, já que há a possibilidade de reduzir problemas a estes sistemas.

Para ser atendido por um servidor de banco, uma pessoa deve esperar em uma fila até que as pessoas a sua frente sejam atendidas. Esse sistema de filas pode ser modelado pelo autômato infinito determinístico ilustrado pela Figura 9, em que $+$ e $-$ são os respectivos eventos de chegada e partida de pessoa da fila.

Figura 9 – Um sistema de filas simples

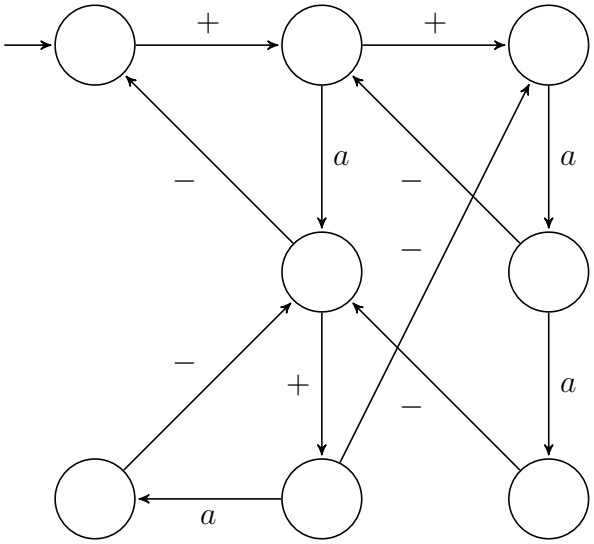


Fonte: CASSANDRAS; LAFORTUNE, 1999.

Na Figura 9 é assumido que a fila de pessoas pode crescer indefinidamente, o que não é possível na realidade. Assumindo que as filas têm um tamanho máximo, faz-se possível modelar os sistemas de filas por meio de autômatos finitos determinísticos (AFDs). Para muitos destes sistemas, além do mais, deseja-se que as filas respeitem uma capacidade máxima, sem que operações de adição sejam permitidas quando ela é alcançada. Outra propriedade frequentemente esperada é a de que nenhum evento de retirada seja possível quando há zero elementos na fila. A garantia de tais restrições é o que configura o problema do produtor-consumidor (MEHMOOD et al., 2011), mas não se restringe a ele.

Um exemplo de sistema de filas é apresentado por Ferreira (2019) e consiste em uma planta de fila de demandas para veículos aéreos não tripulados (VANTs). Nele as demandas são os consumidores, e os VANTs, os servidores. Seja o AFD da Figura 10, em que os eventos $+$, a e $-$ indicam a chegada, atribuição e conclusão de demanda. O modelo representa um sistema de fila de demandas para VANTs, cujas atribuições só podem ser retiradas da fila depois de concluídas e o tamanho da fila é no máximo 2. O AFD torna-se complexo se consideramos a possibilidade de falhas e reatribuição, e determinar se restrições são atendidas fica ainda mais difícil.

Figura 10 – Um sistema de demandas para VANTs



Fonte: Elaborada pelo autor, 2019.

A julgar pela dificuldade de garantir consistências em SEDs, é eminente a relevância de métodos formais para inferências a respeito de propriedades dos AFDs. Antes de formular esses meios matemáticos, duas importantes propriedades dos sistemas de filas são expostas na Seção 5.1.

5.1 PROPRIEDADES DESEJADAS

Como citado para o problema do produtor-consumidor, um sistema de filas geralmente requer que duas propriedades inerentes sejam satisfeitas: que o tamanho máximo de cada fila não possa exceder uma capacidade máxima e não seja possível realizar operações de retirada em uma fila vazia. Isso decorre do fato de que normalmente as filas destes sistemas são representações de espaços físicos, com limitações que devem ser respeitadas nos modelos de SEDs.

O problema do produtor-consumidor decorre da necessidade de sincronizar um *buffer* – uma fila – compartilhado entre processos. Na ciência da computação,

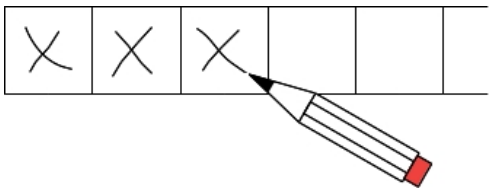
a concorrência de processos sobre os mesmos recursos impõe que alguns artifícios de software e hardware sejam empregados a fim de respeitar as propriedades explanadas por esta seção. Sem eles, o desenvolvimento de aplicações computacionais com memória compartilhada gera defeitos que são manifestados no uso. Um exemplo deste problema é o caso em que dois processos tentam adicionar arquivo no *spool* de impressão de um sistema operacional, o que, se não for sincronizado, possibilita a ambos os processos escreverem sobre a mesma célula do *spool* (TANENBAUM; BOS, 2016). No contexto da automação industrial, há também a necessidade de manter processos de produção e manufatura sincronizados. Máquinas que compartilham filas de recursos precisam atuar de modo consistente.

Apesar de as representações de filas físicas requererem um volume mínimo de zero itens, esta pode não ser uma condição para filas abstratas. A subseção 5.1.1 aborda esses e outros casos, em que é possível reduzir problemas abstraindo a noção de filas, servidores e consumidores.

5.1.1 Redução de problemas

É possível reduzir problemas à validação das propriedades supracitadas. Seja um SED com a seguinte restrição: há dois subconjuntos de eventos E_1 e E_2 tais que, em qualquer cadeia de eventos executada pelo sistema, o número de elementos pertencentes a E_1 menos o número de elementos pertencentes a E_2 deve ser menor ou igual a n. Abstraindo, podemos imaginar que um supervisor anota um símbolo em uma fila toda vez que um evento de E_1 é executado e apaga dela um símbolo quando um evento de E_2 é efetuado, da forma que a Figura 11 ilustra. Pode-se modelar esse SED como um sistema de filas em que os eventos de E_1 e E_2 sejam substituídos respectivamente por operações de adição e retirada de fila. Garantido que a fila tem capacidade máxima de n itens, a restrição é satisfeita, porque, sempre que há x símbolos na fila, foram executados x eventos de E_1 a mais que de E_2.

Figura 11 – Abstração de filas para redução de problemas

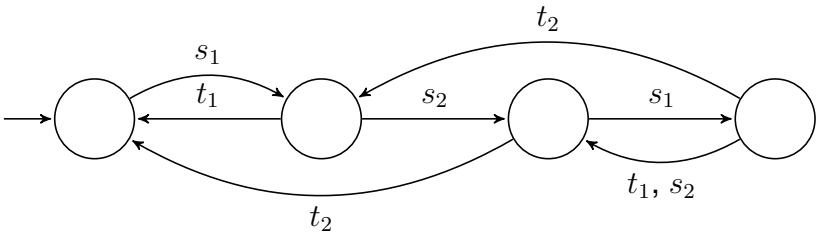


Fonte: Elaborada pelo autor, 2019.

Há outros meios de reduzir problemas de SEDs à garantia das propriedades de sistemas de filas. Para tanto, deve-se encontrar meios de abstraí-los usando os

conceitos que este capítulo apresentou. A exemplo dessas abstrações, seja o autômato da Figura 12, que descreve o funcionamento de uma linha de produção em que há uma esteira e dois robôs com sensores embutidos. Quando uma peça passa pela esteira e chega perto de um dos robôs, o sensor deste detecta a peça e emite sinal para que o robô pegue-a, faça algum trabalho nela e coloque-a em outra esteira. Se o primeiro robô está ocupado, o segundo detectará por seu sensor a peça passando e poderá executar o trabalho. Suponhamos que uma análise criteriosa do sistema foi levantada e originou o AFD ilustrado e identifiquemos os respectivos eventos s_1, s_2, t_1 e t_2 como a detecção de peça pelos sensores do primeiro e segundo robô e a execução de trabalho na peça pelo primeiro e segundo robô.

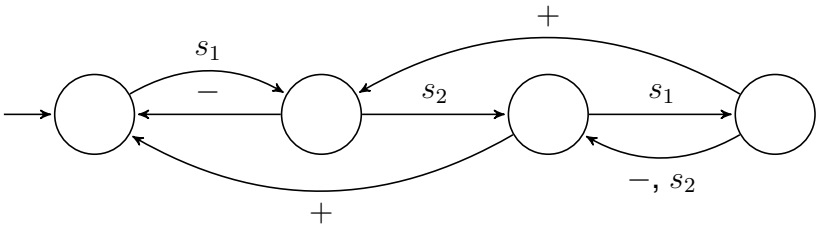
Figura 12 – Autômato para o funcionamento de uma linha de produção simples



Fonte: Elaborada pelo autor, 2019.

Sobre o sistema modelado pela Figura 12, utilizando a abstração por filas, podemos responder à pergunta: há uma carga máxima de trabalho que o segundo robô pode executar a mais que o primeiro? Observando o diagrama, é notável que não. Entretanto, seja uma fila com todos os trabalhos que o segundo robô executa a mais que o primeiro. O sistema de filas da Figura 13 apresenta tal redução, e notemos a existência de um ciclo que pode se repetir indefinidas vezes, adicionando um item à fila por vez. Isso evidencia uma característica da fila: não existe capacidade a limitando. O exemplo é simples, mas ilustra claramente uma redução de problema. Para problemas e SEDs complexos, esse procedimento mostra-se interessante, porque propicia o emprego dos métodos matemáticos do presente trabalho na resolução.

Figura 13 – Autômato da Figura 12 reduzido a um sistema de filas



Fonte: Elaborada pelo autor, 2019.

Especificadas as propriedades acima, resta encontrar formalismos capazes de auxiliar a garantia delas. O capítulo seguinte, 6, propõe e exemplifica teoremas para esse propósito.

6 GARANTIA DE PROPRIEDADES EM SISTEMAS DE FILAS

Posto que é desejável a constatação de certas propriedades referentes a sistema de filas, um grupo de sistemas a eventos discretos (SEDs), este capítulo visa demonstrar a garantia de capacidade máxima e volume mínimo zero destes sistemas. Na visão de Reisig (1993), provas acerca de tais propriedades não são completamente triviais, o que destaca a conveniência auferida com a assistência do provador de teoremas Coq.

Introduzamos algumas notações e definições úteis para nosso propósito. A fim de representar as operações de adição e retirada de fila, utilizemos os respectivos símbolos $+$ e $-$. Seja $G = \langle Q, E, \delta, q_0, Q_m \rangle$ um autômato finito determinístico (AFD) definido conforme a Equação 3.1 em que $E \supseteq \{+, -\}$, queremos provar que o sistema de filas modelado por G satisfaz as duas propriedades citadas no Capítulo 5: que a fila tenha uma capacidade máxima e não seja permitido efetuar a operação de retirada de item quando ela estiver vazia.

Precisamos de uma função $c : E^\star \to \mathbb{Z}$ que conte o número de itens armazenados em uma fila após uma cadeia de eventos $w \in E^\star$. Uma forma de defini-la é

$$c(w) = \begin{cases} c(w') + 1 & \text{se } w = +w' \\ c(w') - 1 & \text{se } w = -w' \\ 0 & \text{senão} \end{cases}$$

Em tal definição, consideramos que o número inicial de itens na fila do sistema é sempre zero. Para sistemas de filas com um número $n_0 \neq 0$ de itens inicialmente dispostos na fila, a quantidade de volume armazenado, após uma cadeia de eventos w, será $c(w)+n_0$. Baseado nas propriedades aritméticas da soma de inteiros, podemos observar para quaisquer w_1 e $w_2 \in E^\star$

$$c(w_1 w_2) = c(w_1) + c(w_2)$$

que será útil para a demonstração dos teoremas a respeito das propriedades que queremos garantir.

De modo a facilitar a leitura dos teoremas, simplifiquemos a notação para a transição estendida de um estado q por uma cadeia de eventos w da seguinte maneira:

$$qw \equiv \hat{\delta}(q, w)$$

Pelo mesmo motivo, definamos a função $t : E \to \mathbb{Z}$, que mapeia os eventos ao incre-

mento que eles causam no tamanho da fila, desta forma:

$$t(e) = \begin{cases} 1 & \text{se } e = + \\ -1 & \text{se } e = - \\ 0 & \text{senão} \end{cases}$$

As Seções 6.1 e 6.2 apresentam duas condições necessárias e suficientes para que um dado sistema de filas tenha capacidade máxima e volume mínimo zero respectivamente. Aplicações dos teoremas são exemplificadas após sua introdução.

6.1 GARANTIA DE CAPACIDADE MÁXIMA

O teorema que esta seção aborda surgiu do seguinte questionamento: de que modo é possível mapear cada estado do sistema ao número máximo de itens na fila que pode haver no estado? Com isso, podemos simplesmente verificar o número mapeado por cada estado e obter o maior deles: a capacidade máxima, como desejamos. Há de se salientar que, nos casos de sistemas que permitem um número indefinidamente grande de volume na fila, esse mapeamento não pode existir. Já os sistemas de filas que respeitam a propriedade da capacidade máxima obrigatoriamente permitem o mapeamento.

Obter intuitivamente o mapeamento dos estados para os tamanhos máximos da fila e avaliar seu corretismo pode ser uma tarefa dispendiosa. Verificando um conjunto de regras, não obstante, a atividade torna-se um pouco mais simples. Se um estado q é mapeado por $f : Q \to \mathbb{Z}$ ao máximo volume que pode haver em q, qualquer transição partindo de q deve respeitar

$$f(qe) \geq f(q) + t(e)$$

para qualquer evento $e \in E$. O Teorema 1 utiliza essa concepção para exprimir uma condição suficiente pela qual o SED modelado por G tenha uma capacidade máxima.

Teorema 1. *Existe uma função $f : Q \to \mathbb{Z}$ e um inteiro n tal que*

$$f(q_0) = n_0 \wedge [(\forall q \in Q)(\forall e \in E), f(qe) \geq f(q) + t(e) \wedge f(q) \leq n]$$

em que $n_0 \in \mathbb{Z}$ é o número de itens inicial na fila, se e somente se a fila do sistema modelado por G tem sempre no máximo n itens.

Devemos demonstrar a validade do teorema. Provemos antes o lema que segue, de utilidade para esse fim.

Lema 1. *Para toda função $f : Q \to \mathbb{Z}$, é válido que*

$$f(q_0) = n_0 \wedge [(\forall q \in Q)(\forall e \in E), f(qe) \geq f(q) + t(e)] \\ \Rightarrow \forall w \in L(G), n_0 + c(w) \leq f(q_0 w)$$

Demonstração. A prova deste lema será por indução em w, tendo como base

$$n_0 + c(\varepsilon) \leq f(q_0 \varepsilon)$$

o que é verdade, pois $c(\varepsilon) = 0$ e é pressuposto $f(q_0) = n_0$.

Para o passo indutivo, temos de provar que, para todo evento $u \in E$, se wu é gerada por G, então

$$n_0 + c(wu) = n_0 + c(w) + t(u) \leq f(q_0 wu)$$

ou

$$n_0 + c(w) \leq f(q_0 wu) - t(u) \tag{6.1}$$

é válido a partir da hipótese de indução.

Sabemos que, se wu é gerada por G, então o prefixo w também é. Diante disso

$$n_0 + c(w) \leq f(q_0 w) \tag{6.2}$$

é obtido da hipótese de indução.

Com base na hipótese do lema, obtemos

$$f(q_0 w) \leq f(q_0 wu) - t(u) \tag{6.3}$$

A partir das Equações 6.2 e 6.3, verifica-se

$$n_0 + c(w) \leq f(q_0 w) \leq f(q_0 wu) - t(u)$$

demonstrando a Equação 6.1, como queríamos. □

Dado que o presente relatório é apenas parte de um trabalho de conclusão de curso, justifica-se o caráter parcial que assumirá a prova do Teorema 1. Nesse sentido, a demonstração que segue é a respeito da condição unidirecional da esquerda para a direita.

Demonstração. Aplicando o Lema 1 na hipótese do teorema, obtemos

$$\forall w \in L(G), n_0 + c(w) \leq f(q_0 w)$$

Da hipótese também temos

$$f(q_0 w) \leq n$$

Portanto

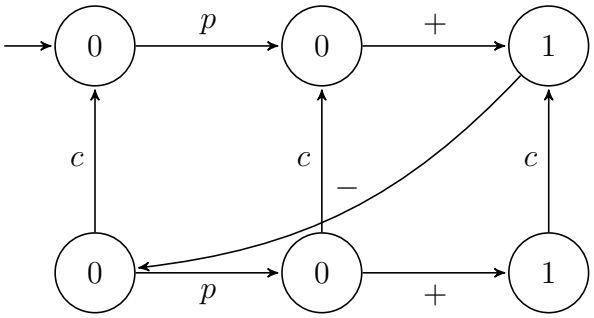
$$n_0 + c(w) \leq f(q_0 w) \leq n$$

como desejávamos demonstrar. □

A demonstração do Lema 1 e Teorema 1 foram auxiliadas pelo Coq, no qual a formulação de AFDs seguiu o exposto na Subseção 3.1.7. Ao ser transcrita a prova do assistente para este trabalho, as noções de tipos foram trocadas pelas de conjuntos, com a qual estamos habituados na teoria dos autômatos.

Na figura 14, há um exemplo que ilustra um sistema de produtor e consumidor com $n_0 = 0$. Nela os eventos p e c são, respectivamente, a produção e consumo de item. Seja f_1 uma função cuja imagem está rotulada nos nós da figura, indicando para cada estado o valor mapeado. Como f_1 respeita o Teorema 1, podemos concluir que o sistema modelado permite volume máximo de um item na fila.

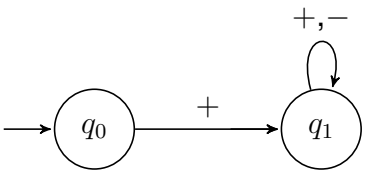
Figura 14 – Exemplo de aplicação do Teorema 1 em um sistema com capacidade máxima 1



Fonte: Elaborada pelo autor, 2019.

Sob outra perspectiva, a figura 15 apresenta um exemplo de AFD modelando um sistema que permite, notavelmente, adição e retirada de um número indefinido de itens da fila. Supondo que tal sistema atenda à propriedade da capacidade máxima, seja $f_2 : \{q_0, q_1\} \to \mathbb{Z}$ qualquer função que mapeie cada estado a um número inteiro de forma que $f_2(q_0) = n_0$ e $f_2(qe) \geq f_2(q) + t(e)$ para todo evento $e \in \{+, -\}$. É fácil notar que f_2 não existe, pois, como $q_1 + = q_1$, podemos obter o absurdo: $f(q_1) \geq f(q_1) + t(+)$. Com isso, o Teorema 1 conclui que o sistema não admite capacidade máxima.

Figura 15 – Exemplo de sistema que não tem capacidade máxima definida



Fonte: Elaborada pelo autor, 2019.

Além de provar que determinados sistemas de filas cumprem ou não com a especificação de uma fila com volume máximo, muitas vezes quer-se atestar a impossibilidade de efetuar a operação de retirada de fila vazia. É sobre isso que trata a Seção 6.2.

6.2 GARANTIA DE VOLUME MÍNIMO ZERO

Todo sistema de filas especificado com base em Cassandras e Lafortune (2008) deve impedir a remoção de itens de filas quando estas forem vazias. Isso é em razão do caráter físico e não abstrato com que se configuram as filas por aquela ótica. Mesmo para filas abstratas, pode ser interessante que esta propriedade seja respeitada. A fim de garantir isso, o Teorema 2 nos possibilita obter resposta à pergunta: a fila tem volume mínimo de zero itens?

Teorema 2. *Existe uma função* $f : Q \to \mathbb{Z}$ *tal que*

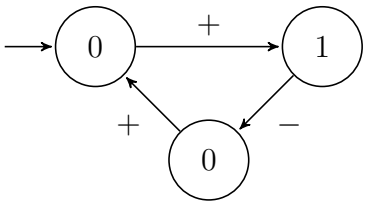
$$f(q_0) = n_0 \wedge [(\forall q \in Q)(\forall e \in E), f(qe) \leq f(q) + t(e) \wedge f(q) \geq 0]$$

se e somente se a fila do sistema modelado por G *nunca tem volume negativo.*

A ideia intuitiva que originou o teorema é semelhante à com que se concebeu o Teorema 1. Neste caso, o mapeamento é feito de cada estado q ao número mínimo de itens que podem haver na fila ao se processar qualquer cadeia de eventos fazendo o autômato transicionar de q_0 a q. Mais uma vez, se é impossível realizar o mapeamento, o sistema não cumpre com a propriedade.

Considerando o autômato da figura 16, verificamos que o sistema nela representado tem uma fila que respeita esta propriedade, embora não garanta capacidade máxima. Tomando a função f_3 rotulada nos nós da figura, pode-se notar que ela respeita o Teorema 2, evidenciando que tal SED foi modelado de modo a não permitir o evento $-$ nos momentos em que a fila está vazia.

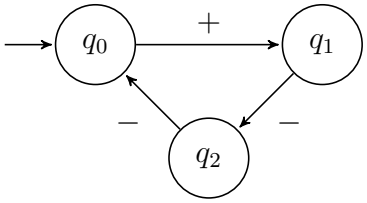
Figura 16 – Exemplo de sistema que atende à propriedade de volume mínimo zero



Fonte: Elaborada pelo autor, 2019.

Como exemplo de aplicação do Teorema 2 para um caso de não atendimento a esta propriedade, seja o AFD da figura 17. Supondo que o sistema satisfaça a restrição de volume mínimo zero, deve haver uma função $f_4 : \{q_0, q_1, q_2\} \to \mathbb{Z}$ respeitando $f(q_0) = n_0$ e $f_4(qe) \leq f_4(q) + t(e)$ para todo evento $e \in \{+, -\}$. Dessa suposição, obtemos $f_4(q_1) \leq f_4(q_1-) - 1 \leq f_4(q_1 - -) - 2 \leq f_4(q_1 - -+) - 1$, um absurdo, uma vez que $q_1 - -+ = q_1$. Dessarte, o Teorema 2 implica que, no sistema modelado, há a possibilidade de executar operações de retirada de fila mesmo quando ela está vazia.

Figura 17 – Exemplo de sistema que permite operações de retirada de fila vazia



Fonte: Elaborada pelo autor, 2019.

Quando aplicamos o Teorema 2 no autômato da figura 14, notamos que a própria função rotulada nos nós do diagrama respeita as regras da condição apresentada nesta seção. Em razão desse fato, o tamanho da fila do sistema ilustrado é sempre algum inteiro do intervalo $[0, 1]$.

7 CONSIDERAÇÕES PARCIAIS

Este trabalho apresentou os principais conceitos relacionados à modelagem de sistemas a eventos discretos (SEDs) por meio de autômatos finitos determinísticos (AFDs), bem como os relativos a sistemas de filas. As duas restrições do problema do produtor-consumidor foram abordadas, originando métodos formais a fim de garantir as propriedades da capacidade máxima e volume mínimo zero em sistemas de filas. Um teorema foi provado mediante o assistente de provas Coq, enquanto outro será provado na continuação deste trabalho de conclusão de curso.

Como resultado de produção do trabalho, foram escritas 237 linhas de código Gallina na IDE do Coq. O processo passou por uma formalização inicial, que foi simplificada gradativamente. O estado parcial da implementação é uma interface mais bem estruturada e organizada.

Entre as dificuldades encontradas durante a produção deste trabalho, pode-se citar a procura de aplicações realistas de SEDs e sistemas de filas. Os exemplos citados neste trabalho são demasiados simples, não correspondendo a sistemas reais da automação industrial. Outro impasse surgiu nos estudos bibliográficos acerca dos assistentes de provas, uma vez que há muitas teorias que sustentam tais artefatos.

Para a continuidade do presente trabalho, novos teoremas e problemas poderão ser formulados. A generalização dos métodos formais para autômatos finitos não determinísticos também possivelmente será feita, assim como a especificação e verificação de algoritmos de *model-checking* baseados nos teoremas.

Com respeito aos desenvolvimentos feitos no Coq, as formulações e provas relativas aos teoremas expostos neste trabalho encontram-se na página do Github do autor, acessíveis em:

`https://github.com/fil1pe/tcc`

REFERÊNCIAS

ATHALYE, A. **CoqIOA: a formalization of IO automata in the Coq proof assistant**. Tese (Doutorado) — Massachusetts Institute of Technology, 2017.

BARRAS, B. et al. **The Coq proof assistant reference manual**. Rocquencourt: INRIA, 1999. v. 6.

BRAIBANT, T.; POUS, D. Deciding kleene algebras in Coq. **arXiv preprint arXiv:1105.4537**, 2011.

CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to discrete event systems**. 2. ed. New York: Springer Science+Business Media, 2008.

DOCZKAL, C.; KAISER, J. O.; SMOLKA, G. A constructive theory of regular languages in coq. In: SPRINGER. **International Conference on Certified Programs and Proofs**. [S.l.], 2013. p. 82–97.

FERREIRA, A. B. de H. **Míni Aurélio: o dicionário da língua portuguesa**. Curitiba: Editora Positivo, 2010.

FERREIRA, V. L. **Coordenação de múltiplas aeronaves não tripuladas baseada na alocação dinâmica de tarefas associadas à logística de entregas** — Universidade do Estado de Santa Catarina, 2019. Disponível em: <http://sistemabu.udesc.br/pergamumweb/vinculos/000071/0000714c.pdf>. Acesso em: 16 out. 2019.

GONTHIER, G. The four color theorem in coq. In: **TYPES 2004 International Workshop**. [S.l.: s.n.], 2004.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. **Introduction to automata theory, languages, and computation**. Boston: Pearson/Addison Wesley, 2007.

LAFORTUNE, S. Discrete event systems: Modeling, observation, and control. **Annual Review of Control, Robotics, and Autonomous Systems**, Annual Reviews, v. 2, p. 141–159, 2019. Disponível em: <https://www.annualreviews.org/doi/abs/10.1146/annurev-control-053018-023659>. Acesso em: 18 out. 2019.

LOPES, Y. K. et al. Finite automata as an information model for manufacturing execution system and supervisory control integration. **IFAC Proceedings Volumes**, Elsevier, v. 45, n. 6, p. 212–217, 2012.

LUO, Z. **Computation and reasoning: a type theory for computer science**. Oxford New York: Clarendon Press Oxford University Press, 1994.

MEHMOOD, S. N. et al. Implementation and experimentation of producer-consumer synchronization problem. **International Journal of Computer Applications**, International Journal of Computer Applications, v. 975, n. 8887, p. 32–37, 2011.

MENEZES, P. F. B. **Linguagens formais e autômatos**. Porto Alegre: Sagra-Luzzatto, 2005.

MURATA, T. Petri nets: Properties, analysis and applications. **Proceedings of the IEEE**, IEEE, v. 77, n. 4, p. 541–580, 1989.

REISIG, W. Specification, modelling and verification of concurrent discrete event systems. In: IET. **IEE Colloquium on Discrete Event Systems: A New Challenge for Intelligent Control Systems**. [S.l.], 1993. p. 4–1.

TANENBAUM, A. S.; BOS, H. **Sistemas operacionais modernos**. 4. ed. São Paulo: Pearson, 2016.

TVERETINA, O. Towards the safety verification of real-time systems with the Coq proof assistant. **Journal of Automation Mobile Robotics and Intelligent Systems**, v. 3, p. 30–32, 2009.