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

Filipe Ramos
Especificação e prova de propriedade acerca de autômatos finitos determinísticos assistidas por Coq/ Filipe Ramos. – Joinville - SC, 2019-
43 p. : il. (algumas color.) ; 30 cm.

Dra. Karina Girardi Roggia

– Universidade do Estado de Santa Catarina - Udesc, 2019.

1. Tópico 01. 2. Tópico 02. I. Prof. Dr. xxxxx. II. Universidade do Estado de Santa Catarina. III. Centro de Educação do Planalto Norte. IV. identificação xxxx

CDU 02:121:005.7

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

Banca examinadora:

Dra. Karina Girardi Roggia
Instituto Superior Técnico de Lisboa

Dr. Cristiano Damiani Vasconcellos
Universidade Federal de Minas Gerais

Dr. Roberto Silvio Ubertino Rosso Junior
Loughborough University

Joinville - SC, xx de xxxx de 2019

Dedico este trabalho a...

AGRADECIMENTOS

Gostaria de agradecer...

Aqui devem ser colocadas os agradecimentos às pessoas que de alguma forma contribuíram para a realização do trabalho.

“frase”

autor

RESUMO

O resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original. O resumo deve ser precedido da referência do documento, com exceção do resumo inserido no próprio documento. (. . .) As palavras-chave devem figurar logo abaixo do resumo, antecedidas da expressão Palavras-chave:, separadas entre si por ponto e finalizadas também por ponto.

Palavras-chaves: latex, abntex e editoração de texto.

ABSTRACT

Resumo em inglês

Key-words: latex, abntex e text editoration.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE SIGLAS E ABREVIATURAS

AFD Autômato finito 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. 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.

Sistemas que são orientados a eventos discretos e assíncronos podem ser modelados por formalismos da matemática discreta, entre eles os autômatos determinísticos, as redes de Petri, *timed models* e cadeias de Markov. São exemplos de SEDs vários sistemas de filas, computacionais, de comunicação, manufatura, tráfego, banco de dados, telefonia, etc. Essas classes não são mutuamente exclusivas, no sentido de haver sistemas de filas que são de manufatura, por exemplo. Podemos modelar SEDs com as características dos sistemas de filas a fim de representar plantas industriais, como linhas de produção complexas. Os AFDs, pois, mostram-se como meios eficientes a esse fim, já que permitem análise por meio de formalismos, o que inclui averiguar a satisfação de propriedades.

A demonstração de propriedades a respeito de AFDs pode ser uma tarefa pouco trivial. Por isso, o emprego de assistentes interativos de provas matemáticas é útil a fim de obter demonstrações válidas. Ademais, tais assistentes reúnem meios formais e computacionais que possibilitam a verificação dos passos da demonstração, auferindo rigor. Entre os assistentes de provas, o Coq destaca-se por ter comunidade extensa e qualidades como a construção de tipos dependentes, motivos pelos quais o presente trabalhou optou por empregá-lo.

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, devemos ser capazes de identificar as eventuais falhas decorrentes da concorrência por recursos distribuídos em filas, justificando o 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 suas aplicações.

Com a finalidade de atingir os objetivos supracitados, o presente trabalho é estruturados destarte. Primeiramente, apresenta-se uma breve noção sobre assistentes de provas e elencam-se os artifícios do Coq mais importantes para o trabalho. No capítulo 3, são introduzidos os SEDs, assim como sua modelagem por AFDs. O capítulo seguinte, 4, apresenta os sistemas de filas, problemas relacionados e as propriedades mais importantes acerca desses SEDs. Em seguida, o capítulo 5 demonstra como constatar se sistemas de filas atendem às propriedades. Por fim, são expressas as considerações finais e, depois, as referências bibliográficas.

2 ASSISTENTES DE PROVAS

Um assistente de provas interativo é um artifício de software que auxilia no processo de prova matemática. 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).

A seção 2.1 introduz o assistente de provas empregado neste trabalho de conclusão de curso. A escolha do Coq se deve às características que seguem.

2.1 COQ

Coq é um dos assistentes de provas mais utilizados atualmente, permitindo a construção de tipos dependentes e polimórficos, lógica de alta ordem, 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 devemos nos 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:

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

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:

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

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`:

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

Já o comando `Record` aproxima o usuário das linguagens de programação comuns, ao definir estruturas parecidas com as quais desenvolvemos nelas:

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

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 que visualizamos é a combinação do CCI com a linguagem de especificação Ganilla, bastante intuitivos. Para provar teoremas e lemas no Coq, além de definir os tipos que usaremos 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 os elementos de uma igualdade são construídos diferentemente, 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, provemos o teorema

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

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

```
Proof. intros n.
```

Agora, iniciemos 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, temos `n` $=$ `S n'`, sendo `S x` a função sucessor de `x` : `nat`. O objetivo atual é

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

Simplificando com `simpl`, obtemos

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

Agora podemos 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 queríamos demonstrar.

O capítulo seguinte, 3, conceitua os sistemas a eventos discretos e autômatos finitos determinísticos, objetos das demonstrações do capítulo 5. Essas provas foram obtidas e validadas no ambiente do assistente Coq.

3 SISTEMAS A EVENTOS DISCRETOS

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

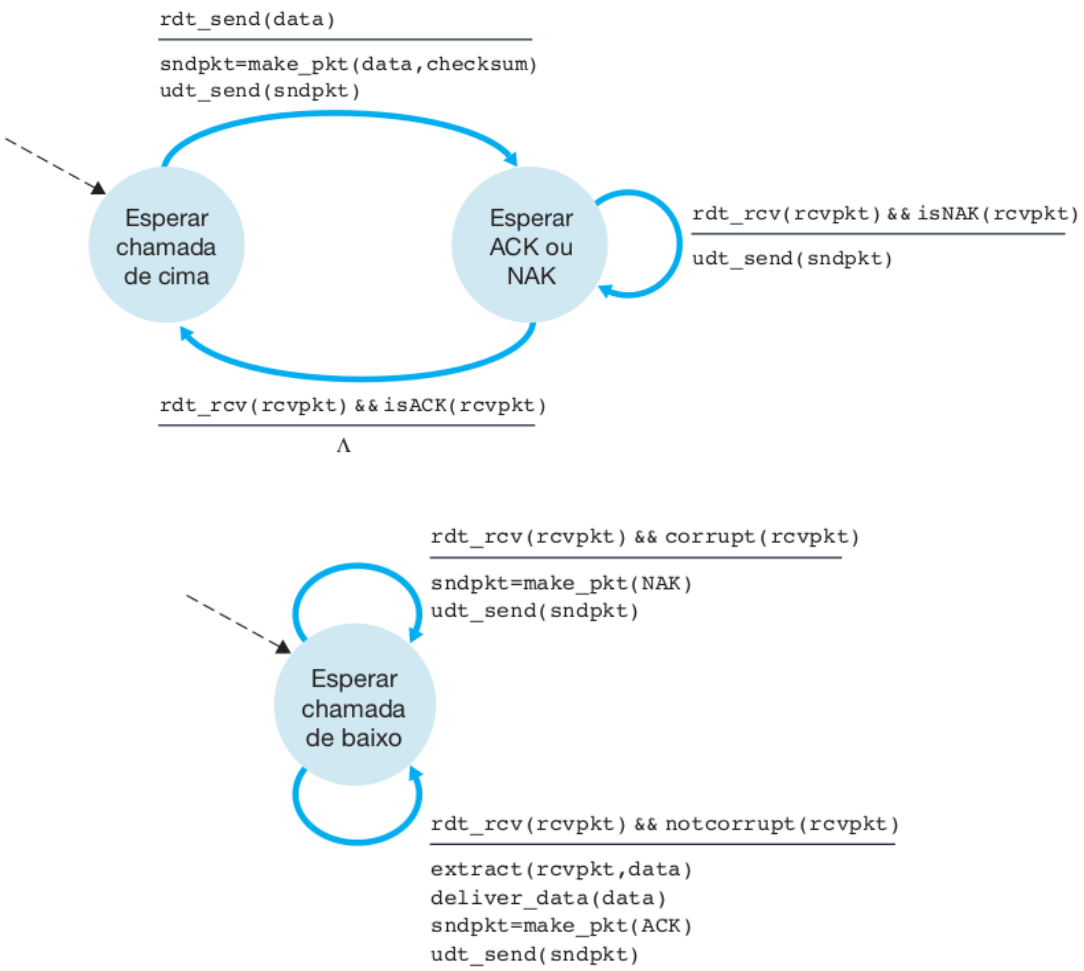
> Sistemas a eventos discretos (SEDs) são sistemas dinâmicos com duas características definidoras: seus espaços de estados são discretos e potencialmente infinitos, e sua dinâmica é orientada a eventos, em vez de tempo. (...) Eventos que ocorrem de forma assíncrona (em geral) causam um salto no espaço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

Cassandras e Lafortune (1999) 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, 1999, 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 (1999) 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 4 explanará essa e outras situações em que podemos abstrair a noção de filas.

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



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 respectiva-

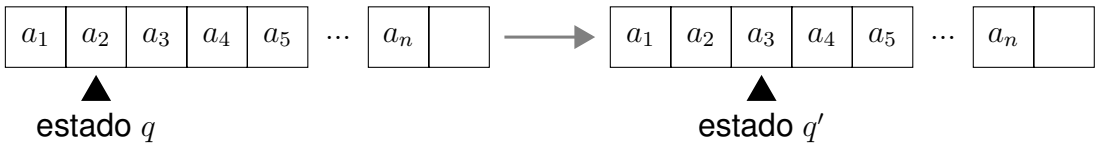
mente e permitem modelar atividades paralelas, protocolos de comunicação, sistemas de produtor-consumidor com prioridade, etc. (MURATA, 1989) Este trabalho de conclusão de concurso, todavia, tem enfoque na representação por autômatos finitos determinísticos. A seção 3.1, que segue, introduz os principais 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 da 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). 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, 1999).

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 (1999).

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 $\Gamma_G : Q \rightarrow 2^E$, 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:

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

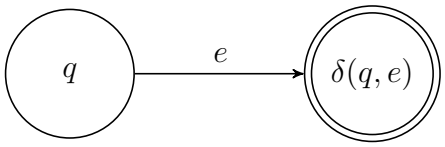
e terá valor indefinido quando $\delta(q, e)$ assim for.

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 seguinte, 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.

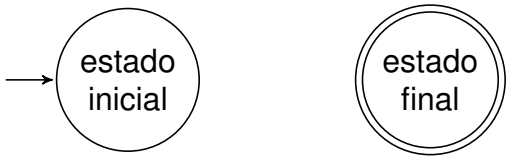
Figura 3 – Representação da transição de estados em um diagrama



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 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

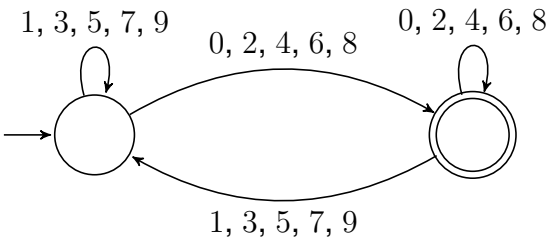
A linguagem marcada por um AFD G é o conjunto

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

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, 1999). 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 seguinte, 3.1.4.

3.1.4 Linguagem gerada e bloqueios

A linguagem gerada por um AFD G é 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, 1999).

3.1.5 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 impreterível 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:

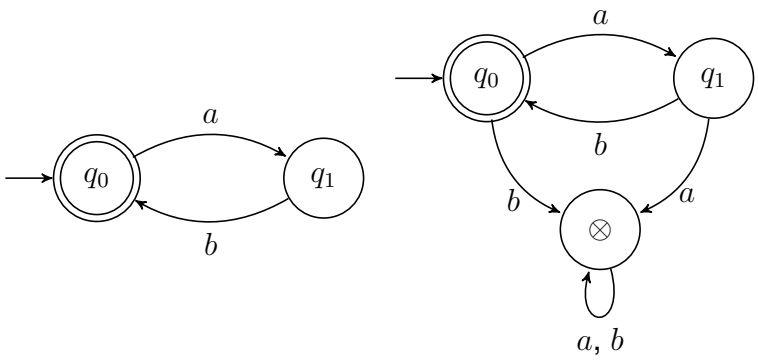
$$L'(G') = \{w \in E^\star \mid \hat{\delta}(q_0, w) \in Q \wedge \hat{\delta}(q_0, w) \neq \otimes\} \tag{3.2}$$

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.

3.1.6 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, podemos construir os 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

$$\delta(q, e) = \Big\{ q' \quad \text{se } \texttt{transition } q \ e = \texttt{proper\_state } q'$$

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

```
Fixpoint 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, determinemos 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, construamos a função `is_marked`:

```
Fixpoint 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 $\langle\{q_1, q_2, ..., q_n\}, \{e_1, e_2, ..., e_m\}, \delta, q$ 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

$$\texttt{extended\_transition g (proper\_state initial\_state) w} \neq \texttt{sink\_state}$$

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`, podemos notar

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

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

4 PROBLEMAS DE FILAS

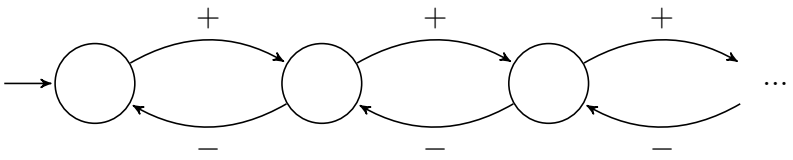
Os sistemas de filas constituem importante classe de sistemas a eventos discretos (SEDs) em que entidades ou objetos devem esperar para obter um determinado recurso. Há três elementos básicos que compõem esses sistemas:

- 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, 1999). 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 8, em que $+$ e $-$ são os respectivos eventos de chegada e partida de pessoa da fila.

Figura 8 – Um sistema de filas simples



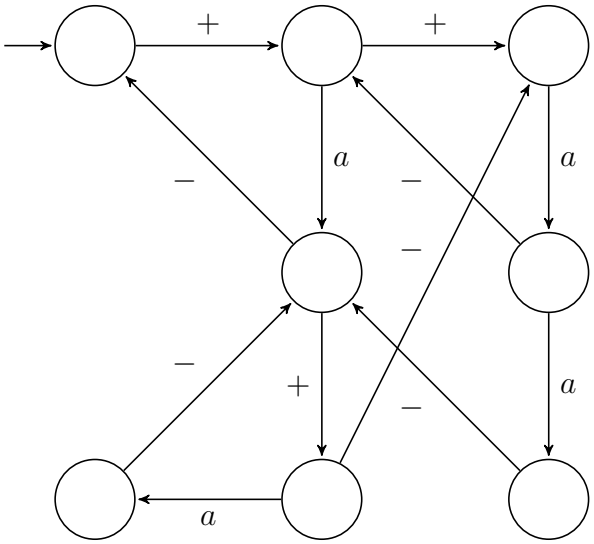
Fonte: CASSANDRAS; LAFORTUNE, 1999.

Na figura 8 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 9, 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 9 – 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 próxima seção, 4.1.

4.1 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.

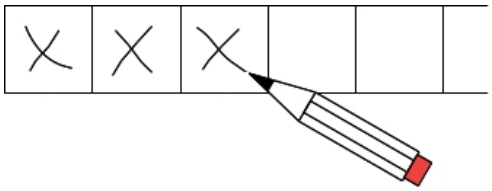
O problema do produtor-consumidor decorre da necessidade de sincronizar um *buffer* – uma fila – compartilhado entre processos. Na ciência da computação, a concorrência de processos sobre os mesmos recursos impõe que alguns artifícios de software e hardware sejam empregados a fim de respeitar as propriedades 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 a seguir aborda esses e outros casos, em que é possível reduzir problemas abstraindo a noção de filas, servidores e consumidores.

4.1.1 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, o número de eventos pertencentes a E_1 menos o número de eventos 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 10 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.

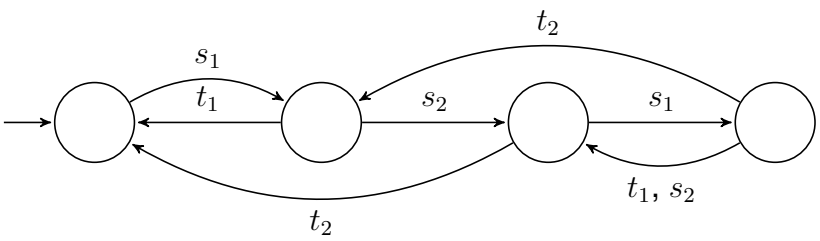
Figura 10 – Abstração de filas para redução de problemas



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 11, 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 11 – Autômato para o funcionamento de uma linha de produção simples

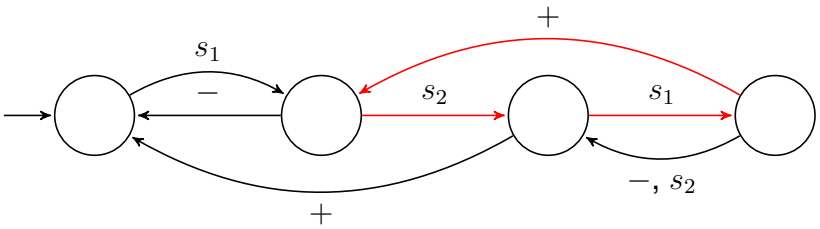


Fonte: Elaborada pelo autor, 2019.

Sobre o sistema modelado pela figura 11, 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? Seja, então, uma fila com todos os trabalhos que o segundo robô executa a mais que o primeiro. O sistema de filas da figura 12

apresenta tal redução, e notemos, pelo circuito destacado em vermelho, 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. Portanto, respondendo à pergunta, não há carga máxima relativa. 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 12 – Autômato da figura 11 reduzido a um sistema de filas



Fonte: Elaborada pelo autor, 2019.

Além de entender a redução de problemas, é interessante observar aplicações que as duas propriedades supracitadas nesta seção proporcionam. É sobre isso que a subseção 4.1.2 versa adiante.

4.1.2 Aplicações

5 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 supracitadas no capítulo 4: 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 $t : E \to \mathbb{Z}$ desta forma:

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

As seções 5.1 e 5.2 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.

5.1 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

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

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

$$n_0 + c(w) \leq f(q_0 wu) - t(u) \tag{5.1}$$

é válido a partir da hipótese de indução.

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

$$n_0 + c(w) \leq f(q_0 w) \tag{5.2}$$

é obtido da hipótese de indução.

Com base na hipótese do lema, obtemos

$$f(q_0 w) \leq f(q_0 wu) - t(u) \tag{5.3}$$

A partir das equações 5.2 e 5.3, verifica-se

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

demonstrando a equação 5.1, 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

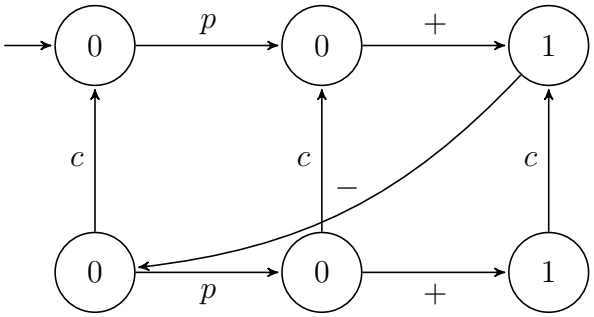
$$n_0 + c(w) \leq f(q_0 w) \leq n$$

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 3.1.6. 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 13, 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.

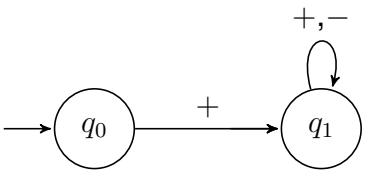
Figura 13 – Exemplo de aplicação do teorema 1 em um sistema com capacidade máxima 1



Fonte: Elaborada pelo autor, 2019.

Sob outra perspectiva, a figura 14 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 14 – 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 5.2, a seguir.

5.2 GARANTIA DE VOLUME MÍNIMO

Todo sistema de filas especificado com base em Cassandras e Lafortune (1999) 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*

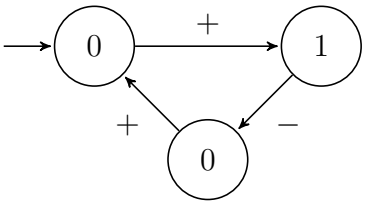
$$f(q_0) = n_0 \wedge [(\forall q \in Q)(\forall e \in E), f(qe) \leq f(q) + t(e) \wedge f(q) \geq 0]$$

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 15, 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.

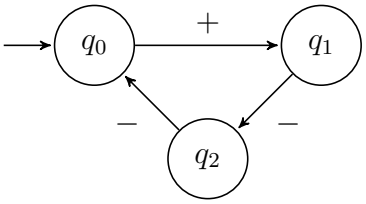
Figura 15 – Exemplo de sistema que atende à propriedade de volume mínimo zero



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 16. 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 16 – 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 13, 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 $[0, 1]$. Os teoremas deste capítulo também podem ser usados nos exemplos do capítulo 4 para efeito de demonstração de aplicação.

6 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. Além disso, foi apresentada a possibilidade de reduzir problemas de SEDs à garantia de propriedades acerca de filas abstratas. Demonstraram-se outras aplicações envolvendo duas restrições de sistemas de filas, que podem ser verificadas por meio de teoremas provados com assistência do Coq. Esse assistente de provas, introduzido no capítulo 2, mostrou-se útil no processo de prova interativa, posteriormente transcrita para o relatório.

Os exemplos e aplicações expostos neste relatório evidenciam a relevância de provar a consistência de sistemas com filas, cujas falhas decorrentes do projeto resultam em problemas para a produção automatizada. Também é eminente a utilidade dos formalismos matemáticos na verificação de características esperadas para um sistema. Após o SED ser projetado e modelado, é possível realizar estudos a respeito de seu funcionamento, em que a matemática é artifício de importância.

Alguns teoremas expostos nesta monografia não foram devidamente provados, posto que isso é trabalho futuro. Ainda, novos problemas, aplicações, exemplos, lemas e teoremas podem vir à tona, o que será acrescido ao trabalho final.

REFERÊNCIAS

BARRAS, B. et al. The coq proof assistant reference manual. **INRIA, version**, v. 6, n. 11, 1999.

CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to discrete event systems**. Boston: Kluwer Academic, 1999.

FERREIRA, A. B. d. 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.

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.

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.