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 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 have been 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

SED Sistema a eventos discretos

SUMÁRIO

1 INTRODUÇÃO

2 PROVAS ASSISTIDAS POR COMPUTADOR

2.1 LÓGICA INTUICIONISTA

2.2 PROVA DIRETA

2.3 TERCEIRO EXCLUÍDO E DECIDIBILIDADE

Uma dificuldade frequente [falar do tanto de perguntas do StackOverflow] dos usuários de assistentes de provas reside no fato de que a lei do terceiro excluído, segundo a qual $P \vee \neg P$ é verdade para qualquer proposição P, não integra a lógica intuicionista da mesma maneira que a lógica clássica.

```
forall x, {P x} + {~ P x}
```

2.3.1 Igualdade decidível

2.3.2 Prova por contradição

2.4 INDUÇÃO MATEMÁTICA

2.5 TÁTICAS DO COQ

3 AUTÔMATOS FINITOS DETERMINÍSTICOS

No âmbito dos SEDs, um autômato finito determinístico (AFD) é uma máquina abstrata cujo funcionamento é descrito por uma dada carga de trabalho ordenada sequencialmente. Ao processar a informação de entrada, codificação sobre as operações requisitadas, o AFD acarreta duas possíveis circunstâncias: a operação corrente não é possível no estado atual da máquina – e portanto o funcionamento é abortado – ou toda operação é possível e feita. Nos dois casos, ao término, o estado do AFD é reiniciado. Além disso o único efeito interno das operações é a transição de estado, a quantidade de estados em que a máquina pode transitar é finita – motivo pelo qual ela é denominada outrossim – e ela não pode estar em mais de um estado simultaneamente, razão por que é determinística. O fato de os eventos, ou operações, ocorrerem em tempo discreto justifica sua modelagem nas teorias dos SEDs. O comportamento de vários desses sistemas pode ser modelado de forma simples, com AFDs.

Alguns estados dos AFDs podem ser marcados para algumas finalidades. Essas máquinas podem proporcionar formalismos reconhecedores de strings; é por isso que às vezes se refere às sequências de operações como palavras.

Haja vista que o presente trabalho objetiva especificar e demonstrar teoremas a respeito de AFDs, é indispensável definir essa classe de máquinas abstratas mediante formalismo. Hopcroft, Motwani e Ullman (2006) formulam AFD como uma quíntupla

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

em que:

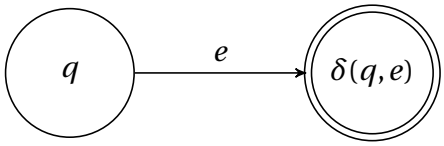
- Q é seu conjunto finito não vazio de estados;
- E é seu conjunto finito de eventos;
- $\delta : Q \times E \nrightarrow Q$ é a função que descreve as transições de estados;
- $q_0 \in Q$ é o estado inicial, a partir do qual a máquina inicia seu trabalho;
- $Q_m \subseteq Q$ é o conjunto de estados marcados.

A definição proposta por Cassandras e Lafortune (2008) inclui uma função de estados ativos $\Gamma : Q \to 2^E$ relacionando as operações possíveis a partir de um estado.

3.1 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 laços que partem de um nó e terminam no mesmo.

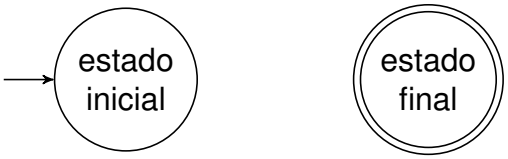
Figura 1 – Representação da transição no diagrama de estados



Fonte: Elaborada pelo autor, 2019.

Nessa classe de diagramas, os estados inicial e marcados 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 2.

Figura 2 – Representação de estados inicial (à esquerda) e marcado (à direita) no 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.

3.2 LINGUAGENS

3.3 FUNÇÃO DE TRANSIÇÃO TOTAL E ESTADO DE RALO

3.4 DEFINIÇÃO FORMAL PARA COQ

Embora amplamente aceitas, as definições matemáticas clássicas de AFD não podem ser diretamente aplicadas no assistente de provas Coq. Ele é, como explana o Capítulo 2, uma implementação de teoria de tipos; assim, é necessário que os elementos integrantes da definição tenham tipos. Ademais, implementações de teoria de conjuntos nessa plataforma não são triviais da forma que se apresentam na lógica interpretada por humanos. É incontrovertível deparar-se com a pergunta: como definir AFDs para estes propósitos e de modo correto?

À primeira vista mostra-se factível utilizar tipos indutivos em vez de implementar estruturas de dados para representar conjuntos. Sendo $\{q_1, q_2, ..., q_k\}$ e $\{e_1, e_2, ..., e_l\}$ os respectivos conjuntos de estados e eventos de um AFD, pode-se construir

```
Inductive Q : Type := q1 | q2 | ... | qk.
```
(3.2)

para os estados e

```
Inductive E : Type := e1 | e2 | ... | el.
```
(3.3)

para os eventos. Contudo, a fim de representar conjuntos finitos genéricos, essas construções não são possíveis. Representá-los como tipos quaisquer também não é sempre uma alternativa conveniente, já que isso permite instanciar tipos contendo infinitos elementos. A prova da decidibilidade de vários [citar exemplos] problemas acerca de AFDs baseia-se na finitude dos conjuntos de estados e eventos, evidenciando a importância de restringir tal atributo.

[Colocar opções que achei do StackOverflow]

Outra opção tange a especificar o recipiente dos estados na forma de lista e atribuir restrições a ela, fazendo o mesmo para os eventos. A biblioteca padrão de listas do Coq implementa uma coleção de funções, definições e notações que é útil para tanto. A implementação possibilita, por exemplo, uma notação semelhante à de pertence ($\in$) da teoria de conjuntos: `In:forall X, X->list X->Prop`. A seguinte construção foi adotada:

```
Record AFD (A B:Type) : Type := {
  Q : list A;
  E : list B;
  transição : A -> B -> A;
```

```
  q0 : A;
  Qm : list A;
  ralo : A;
  transição_correta : forall q e, transição q e <> ralo -> In e E /\ In q
  ↪  Q /\ q <> ralo /\ In (transição q e) Q;
  q0_correto : In q0 Q;
  Qm_correto : forall q, In q Qm -> In q Q;
  ralo_correto : In ralo Q /\ ralo <> q0
}.
```

Restringir a decidibilidade das igualdades de `A` e `B` é importante no sentido de garantir as igualdades decidíveis dos AFDs[1] e permitir aplicar a proposição `transição_correta`, pois verificar se, para quaisquer elementos `q:A` e `e:B`, `transição q e` é igual ao estado de ralo requer

```
forall (a:A), {a = ralo} + {a <> ralo}
```

Mais, essa restrição facilita a prova de alguns dos lemas demonstrados pelo presente trabalho. Ela pode ser assim adicionada ao `Record AFD`:

```
A_decidível : forall (x y:A), {x = y} + {x <> y};
B_decidível : forall (x y:B), {x = y} + {x <> y}
```

A restrição não impede que qualquer AFD definido de acordo com a Equação 3.1 seja escrito nesta formulação, porque os conjuntos de estados e eventos dos AFDs podem ser construídos com os tipos das Equações 3.2 e 3.3, inserindo todos os elementos dessas definições indutivas nas respectivas listas de estados e eventos. A igualdade é decidível para qualquer tipo definido conforme as Equações 3.2 e 3.3. A prova disso é feita por análise de casos da seguinte maneira:

```
Lemma Q_decidível : forall (x y:Q), {x = y} + {x <> y}.
Proof.
  destruct x, y; auto; right; intros; discriminate.
Qed.
```

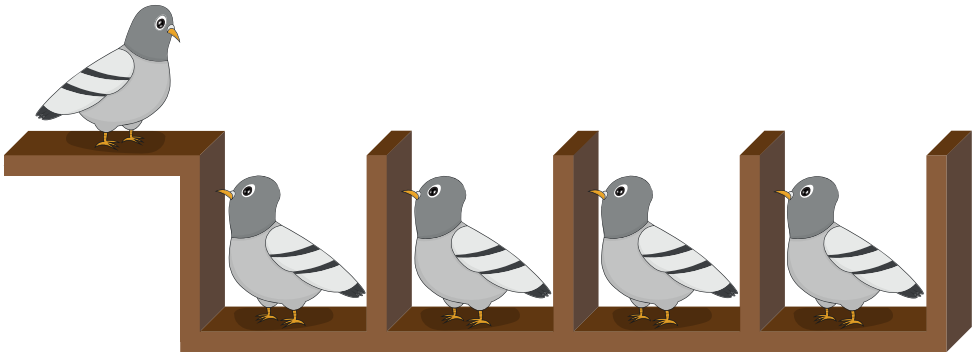
cuja complexidade de tempo é $O(k^2)$, uma vez que a proposição é destruída em k^2 casos. Semelhantemente a complexidade para o tipo `Inductive E` é $O(l^2)$. Em cada caso a tática `auto` automaticamente prova a igualdade, e `right; intros; discriminate` prova a diferença.

[1] Os conjuntos de estados e eventos são finitos; logo, como aborda a Subseção 2.3.1, a igualdade entre seus elementos é decidível.

3.5 LEMAS ÚTEIS

3.5.1 Princípio da casa dos pombos

Figura 3 – Princípio da casa dos pombos



Fonte: Elaborada pelo autor, 2020.

```
Lemma casa_dos_pombos : forall (l1 l2:A),
  (forall x, In x l1 -> In x l2) ->
  length l1 > length l2 ->
  repete l2.
```

De acordo com Öqvist (2017) e Pierce et al. (2019), a prova do lema `casa_dos_pombos` não exige

```
forall (a:A) l, {In a l} + {~ In a l}
```
(3.4)

mas a demonstração de outra forma torna-se muito mais extensa. Para os propósitos deste trabalho, não obstante, o Coq tem, em sua biblioteca padrão, uma prova para a Equação 3.4 – `in_dec` – sob a hipótese de que a igualdade de `A` é decidível. Como a definição de AFD do presente trabalho restringe tal decidibilidade, a demonstração pode ser feita de modo trivial.

3.5.2 Lema do bombeamento

4 SISTEMAS A EVENTOS DISCRETOS

4.1 APLICAÇÕES

4.1.1 Sistemas de filas

4.2 CONTROLE SUPERVISÓRIO

5 TRABALHOS RELACIONADOS

6 TANGIBILIDADE DECIDÍVEL

Ao avaliar a computabilidade de um problema, as complexidades de tempo e espaço não são fatores limitantes.

7 O PROBLEMA DO PRODUTOR-CONSUMIDOR

7.1 CAPACIDADE DE BUFFER

7.2 VOLUME MÍNIMO DE BUFFER

8 COMPLETA INDETECTABILIDADE DECIDÍVEL

```
Definition compl_indetectável := forall w e w', In e G.Eu -> gera (w ++ e
↪  ++ w') -> gera (w ++ w').
```

9 CONSIDERAÇÕES FINAIS

REFERÊNCIAS

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

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. **Introduction to automata theory, languages, and computation**. 3. ed. Boston: Pearson/Addison Wesley, 2006.

ÖQVIST, J. Solution to the Pigeonhole Principle exercise in Software Foundations, without excluded middle. **Github**, 2017. Disponível em: <https://github.com/llbit/Pigeonhole-Principle/>. Acesso em: 3 abr. 2020.

PIERCE, B. C. et al. **Logical Foundations**. 2019. Disponível em: <https://softwarefoundations.cis.upenn.edu/lf-current/index.html>. Acesso em: 8 set. 2019.