Monitoramento de Sistemas Com Autômatos e Sensores: Uma Abordagem Prática

Leandro de Souza Mattos, Humberto Turioni Marinho, Pedro Braga dos Santos Bacellar

Universidade Federal do ABC (UFABC)
Av. dos Estados, 5001 - Bairro Bangu - Santo André - CEP: 09280-560

{mattos.leandro, humberto.turioni, pedro.bacellar}@aluno.ufabc.edu.br

1. Introdução

De acordo com SLOSS [2], um dos pilares da disciplina de engenharia de confiabilidade (SRE) é o monitoramento de sistemas com o uso de alertas. Estes são sistemas automatizados que sinalizam anomalias em sistemas digitais, deixando assim os responsáveis cientes para o tratamento de um cenário onde um problema pode ocorrer ou já está ocorrendo.

O funcionamento básico desses sistemas é baseado em autômatos, ou máquinas de estados finitas. Como definido no trabalho de SIPSER [1], autômatos são constituídos de estados, transições e símbolos de entrada. Conforme um autômato lê um determinado símbolo de entrada ele transita de um estado para outro. A depender da sequência de símbolos recebida, ele pode então rejeitar ou aceitar a cadeia de símbolos de entrada caso o estado em que se encontre ao terminar de processar a cadeia seja considerado um estado final. Formalmente, um autômato é definido como uma quíntupla $M = (\Sigma, Q, \delta, q0 , F)$, onde Σ é o alfabeto de símbolos reconhecido pelo autômato, Q é o conjunto de estados, δ é a função de transição parcial (que gera as transições entre os estados), q0 é o estado inicial e F é o conjunto de estados finais, um exemplo está abaixo demonstrado na Figura 1.

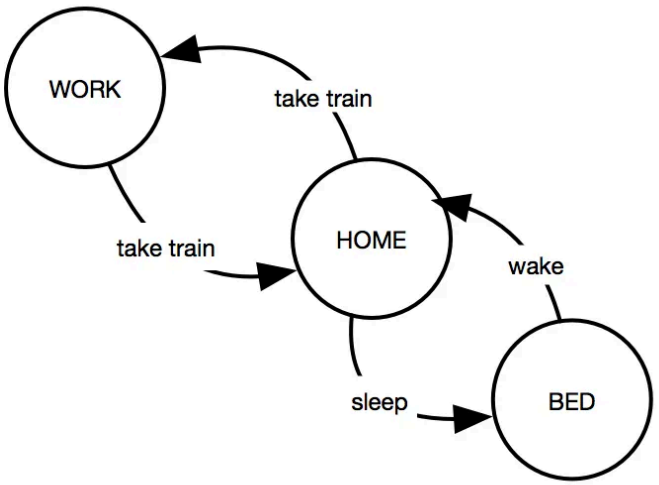

Figura 1: Exemplo de autômato do dia-a-dia

Com o auxílio da biblioteca "automathon" do Python [3], este trabalho investiga o uso de autômatos no monitoramento de sistemas que admitem serem monitorados por sensores e que possuam um critério claro de comportamento anômalo.

2. Descrição do Problema

Um tipo de sistema de monitoramento que faz extenso uso de sistemas de alerta é o de monitoramento de Interfaces de Programação de Aplicações (APIs). No presente estudo, oferecemos um exemplo de como um autômato genérico pode ser utilizado na gestão de erros identificados em uma API, com especial atenção à sua capacidade de emitir alertas na incidência de mais de quatro erros num intervalo de um minuto, ao longo de uma duração total de três minutos.

Definimos uma API e o sistema de monitoramento como os seguintes requerimentos:

1. A condição considerada como falha na API é a de mais do que 4 erros por minuto. Três ou menos erros por minuto são condições normais de operação.
2. Se por três minutos consecutivos (3 ciclos) a condição de falha for detectada, então um alerta deve ser emitido e o sistema se encontrará em um estado de alerta ("**alert**"). Isso significa que se no primeiro ciclo a condição de falha ocorrer, nenhum alerta é emitido. Nesse caso, o sistema fica em uma condição **pendente**. Apenas se o sensor continuar informando a condição de falha por mais dois ciclos (2 minutos) é que um alerta é disparado.
3. Se o sensor não receber dados da API, considera-se que esse é um cenário sem falhas e o sistema se encontrará em um estado de "**no_data**".
4. Se ocorrer um erro no sensor, o sistema ficará no estado de "**error**", mas que também é considerado um cenário sem falhas na API.
5. Do contrário, o estado correspondente é o "**ok**".

A partir dos requerimentos, pode-se começar a construir o autômato. De imediato, extrai-se que os primeiros estados do autômato devem ser "ok", "alert", "no_data" e "error". Há ainda mais 2 estados que correspondem à condição pendente. Como o autômato deve ficar nessa condição por dois ciclos antes de alertar, precisa-se de mais dois estados: "pending1" e "pending2". Já as transições do autômato serão:

1. "s": Sucesso - o sensor nao detectou nenhuma falha
2. "n": No data - o sensor não recebeu dados durante a leitura
3. "f": Falha - a condição de falha foi atingida (mais do que 4 erros por minuto)
4. "e": Error - Houve um erro no sensor

2.1. Descrição formal e desenho do autômato

O autômato é descrito como uma quíntupla: $M = (\Sigma, Q, \delta, q0 , F)$, onde:

$\Sigma = \{s, f, n, e\}$

$Q = \{ok, pending1, pending2, alert, no\_data, error\}$

$q0 = ok$

F = {ok, alert, no_data, error}

δ =

Estado	s	f	n	e
ok	ok	pending1	no_data	error
pending1	ok	pending2	no_data	error
pending2	ok	alert	no_data	error
no_data	ok	pending1	no_data	error
error	ok	pending1	no_data	error
alert	ok	alert	no_data	error

Figura 2 - Tabela de transição do autômato

Com a definição formal, podemos desenhar o autômato:

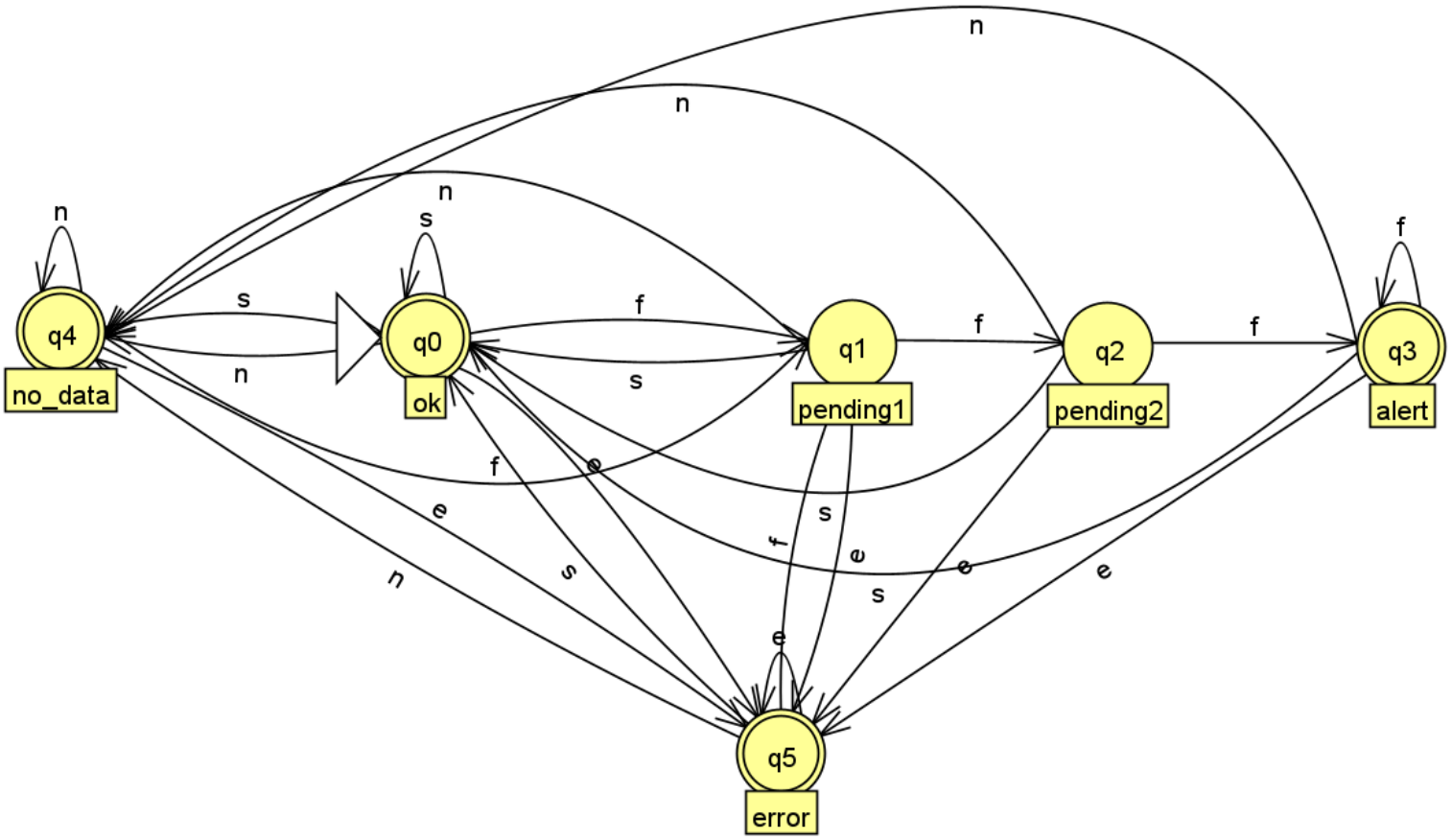

Figura 3 - Diagrama do autômato

2.2. Implementação do autômato

O autômato foi implementado com a biblioteca ‘authomaton’ no Python:

```
Q = {'ok', 'pending1', 'pending2', 'no_data', 'error', 'alert'}
sigma = {'s', 'f', 'n', 'e'}
delta = { 'ok':       {'s': 'ok', 'f': 'pending1', 'n': 'no_data', 'e': 'error'},
          'pending1': {'s': 'ok', 'f': 'pending2', 'n': 'no_data', 'e': 'error'},
          'pending2': {'s': 'ok', 'f': 'alert',    'n': 'no_data', 'e': 'error'},
          'no_data':  {'s': 'ok', 'f': 'pending1', 'n': 'no_data', 'e': 'error'},
          'error':    {'s': 'ok', 'f': 'pending1', 'n': 'no_data', 'e': 'error'},
          'alert':    {'s': 'ok', 'f': 'error',    'n': 'no_data', 'e': 'error'}
        }
initial_state = 'ok'
F = {'ok', 'alert', 'no_data', 'error'}
automata = DFA(Q, sigma, delta, initial_state, F)
```

Pela biblioteca, podemos validar se o autômato é válido:

```
automata.is_valid()
# True
```

Podemos também testar o autômato

```
automata.accept("s")
# True
automata.accept("sf")
# False
automata.accept("sff")
# False
automata.accept("sfff")
# True
automata.accept("sfffen")
# True
```

A biblioteca também cria uma versão NFA do autômato. No entanto, como a versão DFA é completa e mínima, não há como criar uma versão NFA, então ele retorna o mesmo autômato:

```
automata_nfa = automata.get_nfa()
automata.view("afd01_nfa")
display.Image("/content/afd01_nfa.gv.png")
```

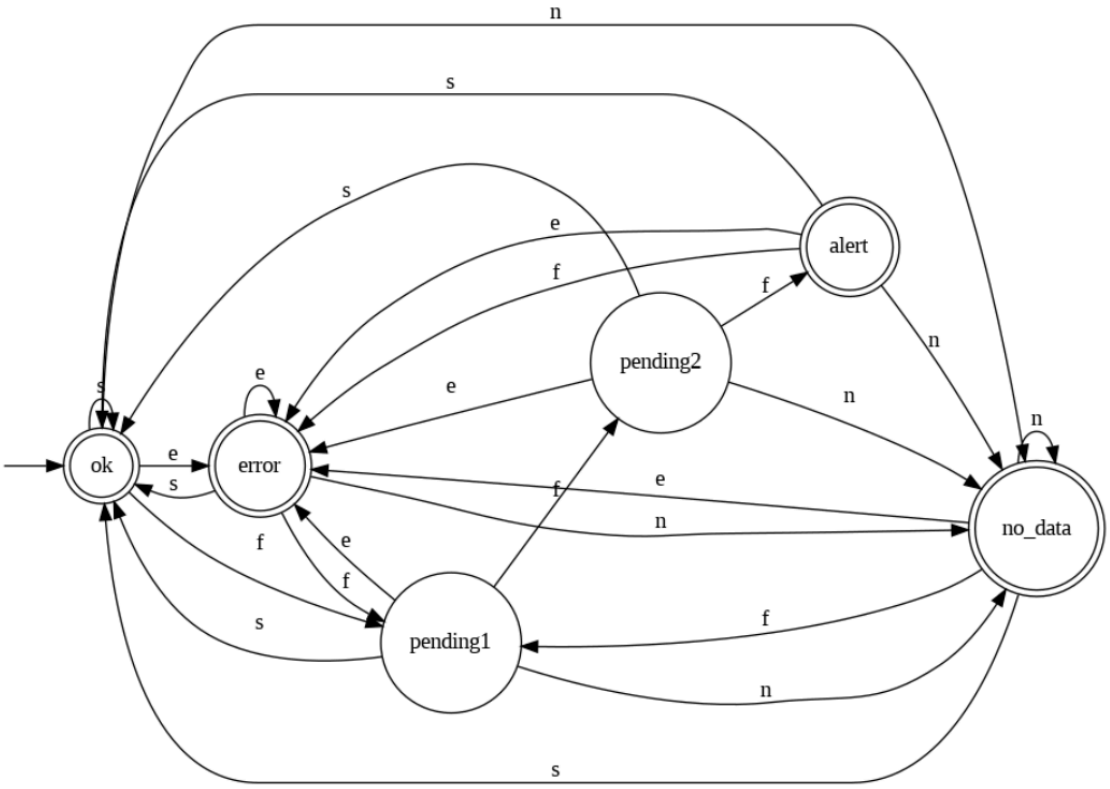

Figura 4 - Autômato NFA

Por fim, podemos pedir para a biblioteca minimizar o autômato, mas o autômato já está minimizado, então o retorno é o mesmo autômato:

```
automata_min = automata_nfa.minimize()
automata.view("afd01_nfa_min")
display.Image("/content/afd01_nfa_min.gv.png")
```

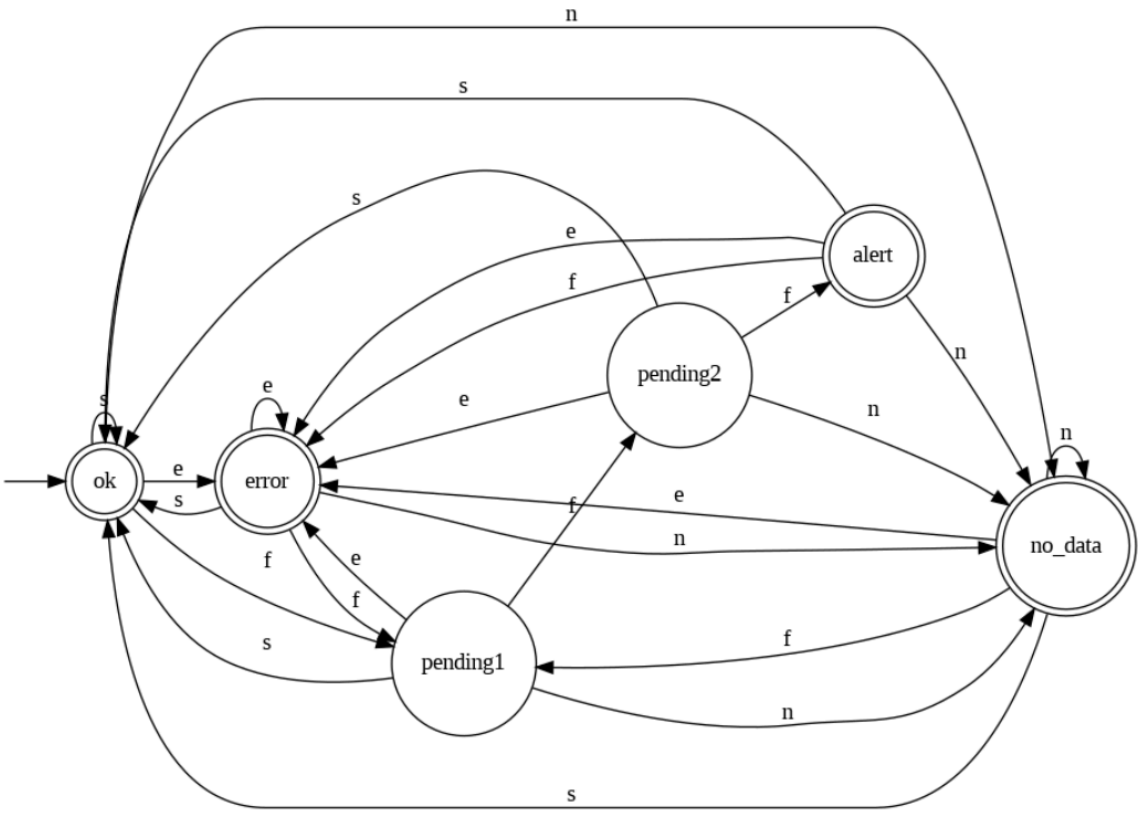

Figura 5 - Autômato minimizado

2.3. Links para os códigos

Link do Google Colab com a implementação em um notebook:
https://colab.research.google.com/drive/1oDlUh_PQqiLyRuLmPH33Lbm6npN9Ztjl?usp=sharing

Link do github com o notebook:

https://github.com/pbacellar/automata_project/

3. Descrição da Aplicação

Conforme mencionado, o autômato apresentado pode ser utilizado para monitorar APIs ou quaisquer sistemas que possam ser medidos por um sensor e possuam uma condição de falha. O autômato poderia controlar, por exemplo, um sistema de monitoramento de tempo de downtime de um servidor. Se o servidor ficar offline por mais de um minuto ao longo de 3 minutos, então um alerta seria emitido.

A grande vantagem do autômato desenvolvido é que ele não emite um alerta imediatamente, assim que a primeira condição de falha é identificada, ele espera 3 ciclos antes de alertar. Dessa forma, evita-se alertas ruidosos ou alertas falsos. Em casos especiais em que se necessita um alerta imediatamente, basta remover os estados pendentes.

4. Considerações Finais

A abordagem apresentada neste artigo oferece uma solução eficaz para o monitoramento de APIs e quaisquer sistemas que podem ser medidos por sensores, utilizando um autômato para automação do processo de detecção e alerta de falhas.

Através do exemplo com uma API, mostrou-se que um autômato determinístico finito, juntamente com um sensor, é capaz de monitorar qualquer sistema que possua alguma condição de falha. Tal autômato representa uma ferramenta valiosa para operadores de sistemas digitais, permitindo resposta rápida a problemas de diversas naturezas e contribui para estabilidade e confiabilidade dos sistemas

Vale destacar o uso real de um autômato similar em sistemas de produção presente na ferramenta Grafana [4]

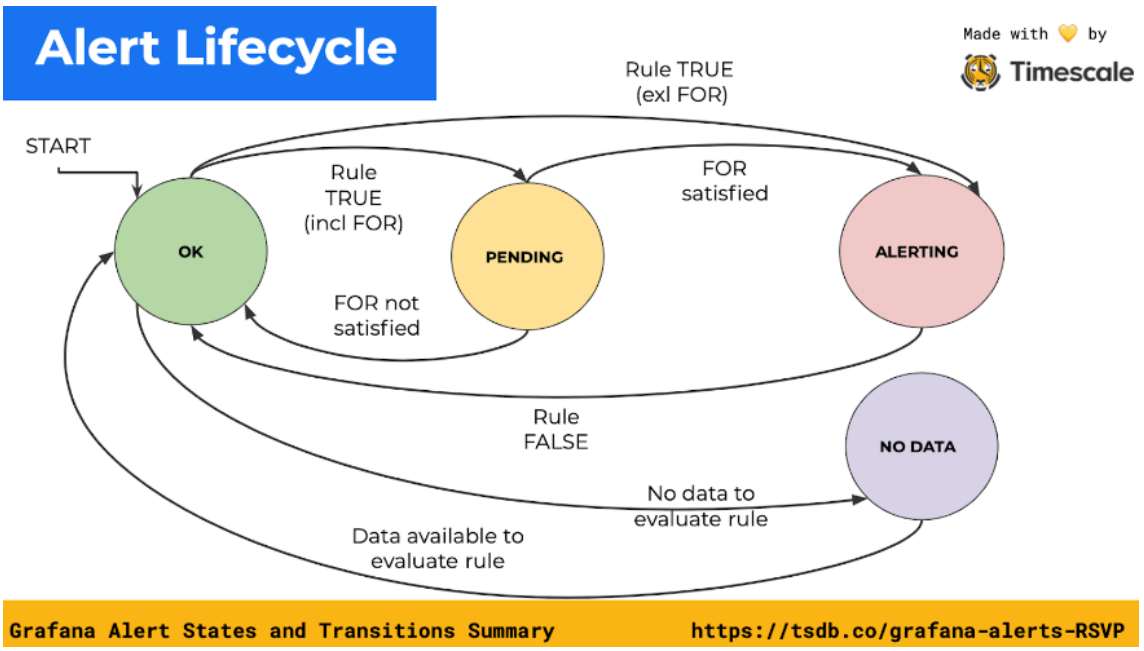

Figura 6 - Autômato do Grafana

5. Referências

1. SIPSER, Michael. Introdução à Teoria da Computação. 1. ed. São Paulo: Editora Cengage, 2005. 488 p. ISBN 978-8522104994.
2. SLOSS, Benjamin Treynor. Introduction. *Site Reliability Engineering*. Disponível em: https://sre.google/sre-book/introduction/. Acesso em: 06 abr. 2024.
3. QUINTERO, R. rohaquinlop/automathon. Disponível em: <https://github.com/rohaquinlop/automathon>. Acesso em: 6 abr. 2024.
4. Timescale. Disponivel em https://www.timescale.com/blog/grafana-101-getting-started-with-alerting-recap-and-resources/. Acessado em 10 abr, 2024.