Detecção de conflitos diretos em redes IoT utilizando o Maude*

Matheus de A. M. Pereira[1], **Daniel Ventura**[1]

[1]Instituto de Informática
Universidade Federal de Goiás (UFG) – Goiânia, GO – Brazil

matheus.marpereira@gmail.com, ventura@ufg.br

Abstract. *In this work we explore the conflict detection in distributed networks problem. We propose a logical concurrency model for the network and specify it in the Maude system, which will be able to simulate the network and detect possible conflicts.*

Resumo. *Neste trabalho exploramos o problema de detecção de conflitos em redes distribuídas utilizando lógica de reescrita. Propomos um modelo lógico e concorrente para a rede e realizamos a sua especificação no software Maude, que será utilizado para simular a rede e detectar possíveis problemas.*

1. Introdução

Sistemas IoT são cada vez mais utilizados no mundo, para diversas finalidades como casas inteligentes ou até mesmo cidades inteligentes, com dispositivos conectados remotamente e que conseguem medir e gerenciar algum aspecto do seu ambiente [Pradeep and Kant 2022]. Estes dispositivos podem ser controlados remotamente por algum aplicativo, ou configurados para agirem automaticamente baseado em certos gatilhos.

Montar uma rede IoT pode ser um grande desafio, pois há aspectos importantes que devem ser considerados para garantir que os dispositivos estão agindo corretamente, que as regras configuradas garantam a integridade e a segurança do sistema [Lee et al. 2017].

Este trabalho apresenta uma modelagem para detecção de conflitos em redes distribuídas utilizando o Maude, um framework de Lógica de Reescrita de alta performance. A seção 2 deste trabalho contém uma introdução à Lógica de Reescrita e ao Maude. A seção 3 contém uma descrição do modelo da nossa rede e seus componentes. A seção 4 contém uma breve descrição para o nosso caso de estudo. A seção 5 explica como foi realizada a especificação no Maude. A seção 6 contém os resultados do nosso trabalho e possíveis trabalhos futuros. A seção 7 contém a nossa conclusão sobre este trabalho.

A especificação completa do modelo no Maude pode ser encontrada no endereço: `https://github.com/mathamp/conflict-detection-maude`.

2. Lógica de Reescrita

Lógica de reescrita é definida como uma *teoria de reescrita sortida* $\mathcal{R} = (\Sigma, E \cup B, R)$, onde Σ é a sintaxe com *sorts* da nossa teoria, E é um conjunto de equações terminante e

*O presente trabalho foi realizado com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) - UNIVERSAL 423212/2021-4

confluente, B um conjunto de axiomas (e.g. associatividade, comutatividade e identidade) e R é um conjunto de regras de reescrita no formato $t \rightarrow t'$ ou no formato condicional: $t \rightarrow t'$ `se` **condição** [Meseguer 2012]. Assim, na representação de um sistema computacional utilizando a lógica de reescrita $\mathcal{R}$, a teoria equacional $(\Sigma, E \cup B)$ representa os estados do sistema como tipos de dados algébricos enquanto o conjunto de regras de reescrita R (possivelmente não-confluente) especifica a dinâmica do sistema.

Devido a natureza não-determinística das regras de reescrita, onde qualquer regra de reescrita compatível com um determinado estado do sistema pode ser aplicada, sistemas concorrentes podem ser naturalmente modelados, devido a incerteza sobre a ordem de execução e comunicação entre os dispositivos. Por exemplo, considere um quarto com dispositivos inteligentes configurados para ligar o aquecedor quando a temperatura do quarto atingir 10ºC e ligar as luzes quando o horário for 18:30, e que, exatamente às 18:30 a temperatura chegou a 10ºC. Devido a natureza concorrente dos dispositivos, não é possível determinar qual ação ocorrerá primeiro: ligar as luzes ou ligar o aquecedor, mas é possível deduzir os cenários que podem ocorrer como um diagrama de transições:

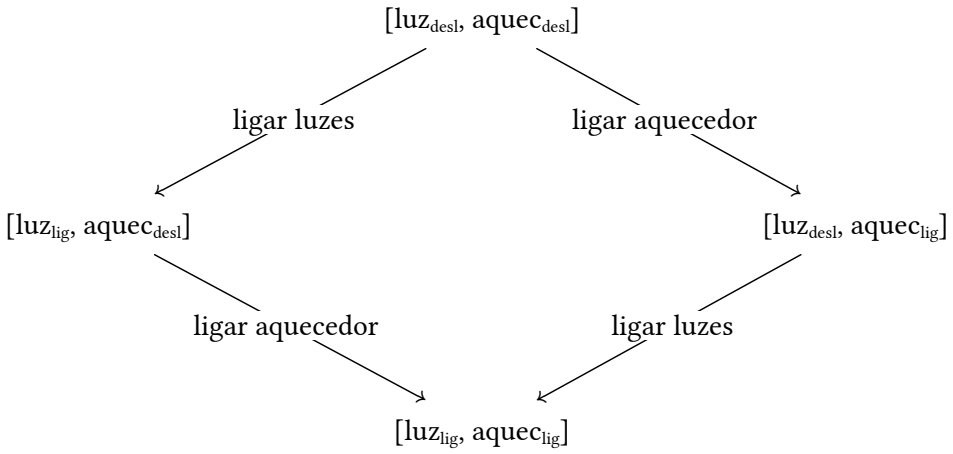

Figura 1. Diagrama de transições

Este é um exemplo de situação confluente, onde todas as possibilidades de transições convergem para um mesmo estado, mas também existem situações não-confluentes, onde a ordem das transições pode resultar em estados divergentes.

2.1. Maude

O software utilizado para a especificação do nosso modelo será o Maude, um framework em Lógica de Reescrita de alta performance. No Maude, a sintaxe Σ para construir sorts e operadores é definida utilizando o comando `op`, as equações E utilizando o comando `eq`, e os axiomas B são definidos entre colchetes como parâmetros no momento da declaração de um construtor. O par $(\Sigma, E \cup B)$ no Maude deve definir funções totais, podendo serem vistas como programas funcionais.

Abaixo apresentamos uma definição de conjuntos de naturais no Maude para introduzir a sua sintaxe:

```
protecting NAT .
sort NatSet .
subsort Nat < NatSet .
```

Primeiro importamos um módulo do prelúdio, chamado `NAT`, que contém a definição dos números naturais. Definimos um sort chamado `NatSet` e especificamos que todo `Nat` também é um `NatSet`, ou seja, um número natural também representa um conjunto unitário de naturais.

```
op empty : -> NatSet [ctor].
op _,_ : NatSet NatSet -> NatSet
            [ctor assoc comm id: empty] .
```

A constante `empty`, representando o conjunto vazio, é definida como um construtor de `NatSet` com uma lista vazia de argumentos. O outro construtor de `NatSet` é definido como um operador infixo, onde `_` indica as posições dos argumentos, listados à esquerda de `->`, no construtor. Também é definido os axiomas de associatividade, comutatividade e `empty` como identidade para este construtor.

```
var N : Nat .
var S : NatSet .

eq N,N = N .

op _in_ : Nat NatSet -> Bool .
eq N in (N,S) = true .
eq N in S = false [owise] .
```

Aqui definimos variáveis para utilizarmos nas nossas equações e operações. A primeira equação traz a propriedade de idempotência para o segundo construtor (Por limitações técnicas, o Maude não consegue lidar com os axiomas `assoc` e `idem` ambos em B). Definimos uma operação para verificar se um elemento está num conjunto através de 2 equações parciais. A primeira equação utiliza *pattern matching* módulo associatividade-comutatividade para verificar se o elemento desejado está no conjunto, caso contrário, a segunda equação é utilizada.

Uma especificação no Maude pode ser dividida em módulos, estamos interessados em 2 tipos de módulos específicos:

- **Módulos funcionais:** Permite a definição de teorias equacionais. Será utilizado para especificar os *sorts* que definem a nossa rede, e algumas funções auxiliares.
- **Módulos de sistema:** Permite a definição de regras de reescrita. Será utilizado para especificar a comunicação entre os dispositivos da rede.

Após especificado o formato da rede e a comunicação entre os dispositivos, uma rede pode ser instanciada e o Maude pode iniciar a busca por estados que representam o acontecimento de conflitos utilizando as regras de reescrita. No Maude, estamos interessados no uso de 2 comandos para a execução das regras de reescrita:

- `rewrite <REDE> .`: Irá aplicar as regras de reescrita de forma determinística. Usado inicialmente para analisar como o Maude está realizando as regras de reescrita e se existem problemas na nossa especificação.

- `search <REDE> =>* <CONFLITO> .`: Verificar se existe alguma ordem de aplicação de regras de reescrita que chega num conflito. Irá explorar todas as possibilidades para aplicação de regras de reescrita, gerando um grafo de busca de estados, que será explorado utilizando uma estratégia de busca em largura. Um detalhe importante no qual deve-se tomar cuidado é não deixar o número de estados crescer subitamente, pois isso impedirá fazer buscas mais profundas em um tempo razoável utilizando o Maude. Note que o crescimento da árvore de busca será determinado pelo número de regras de reescrita aplicáveis em cada estado.

3. Modelo para redes IoT

O modelo é baseado na abordagem proposta por [Al Farooq et al. 2019]. A rede IoT consiste de diversos componentes conectados, que comunicam entre si transmitindo eventos, ações e comandos. Entre esses componentes estão: sensores, controladores, atuadores e os dispositivos IoT.

A forma de comunicação entre esses componentes funciona num modelo "gatilho e ação": cada dispositivo tem um conjunto de gatilhos, quando um gatilho ocorre, este realizará alguma ação. Sensores medem algum parâmetro do ambiente, e quando este atinge algum limiar, o sensor dispara eventos. Controladores escutam por eventos, e podem disparar ações em resposta. Atuadores escutam por ações, e podem disparar comandos em resposta. É importante ressaltar que um sensor pode emitir eventos para vários controladores, controladores podem receber eventos de vários sensores e emitir ações para vários atuadores, e atuadores podem receber ações de vários controladores. Conforme a quantidade de componentes na rede cresce, é possível que haja sobreposições na comunicação entre os componentes, e isto aumenta as chances de ocorrer conflitos.

3.1. Configurações e regras

Uma rede IoT é montada com a finalidade de atender algum objetivo, por exemplo, "Manter a temperatura do quarto agradável", estas são chamadas *regras de ambiente* e devem ser mantidas a todo momento. Se a rede permitir que essas regras sejam quebradas, temos um *conflito de regra de ambiente*.

Os dispositivos da rede podem ser configurados por regras operacionais que ditam o seu comportamento. Um sensor pode ser configurado para observar algum parâmetro da rede e emitir eventos caso o parâmetro observado atinja algum limiar desejado, como por exemplo, "Se a temperatura chegar em 30ºC, emitir o evento **Temperatura Quente**". Um controlador pode ser configurado para escutar por eventos e emitir ações, como por exemplo, "Ao receber o evento **Temperatura Quente**, emitir a ação **Ligar o ar condicionado**". Um atuador pode ser configurado para escutar por ações e emitir comandos, como por exemplo, "Ao receber a ação **Ligar o ar condicionado**, emitir o comando **Ligar** para o ar condicionado".

Dependendo de como a rede for configurada, é possível que um atuador receba 2 ações contraditórias ao mesmo tempo, devido a componentes diferentes que estão configurados para atender objetivos diferentes. Por exemplo, um grupo de componentes está configurados para ligar as luzes do quarto enquanto houver pessoas no quarto, enquanto outro grupo de componentes está configurados para desligar as luzes caso esteja de madrugada. Quando houver pessoas na sala de madrugada, as ações **Ligar luzes** e **Desligar**

luzes serão emitidas, que contradizem entre si. Este tipo de conflito foi identificado como *conflito direto* [Adamczyk and Kliks 2023]. Conflitos de regra de ambiente também são conflitos diretos.

4. Caso para estudo

Consideramos um quarto com alguns dispositivos IoT para controlar automaticamente sua temperatura e iluminação. Para isso foi configurado um conjunto de sensores, controladores e atuadores para o ar condicionado, o aquecedor e as luzes do quarto. A rede foi configurada de forma para ligar e desligar automaticamente o ar condicionado ou o aquecedor quando a temperatura estiver muito quente ou fria, e desligá-los quando a temperatura estiver amena. As luzes ligam e desligam automaticamente baseadas na luminosidade do quarto, e ficam desligadas durante a madrugada. A figura 2 mostra um diagrama que representa como os componentes estão conectados. Para mais detalhes, veja o Apêndice A.

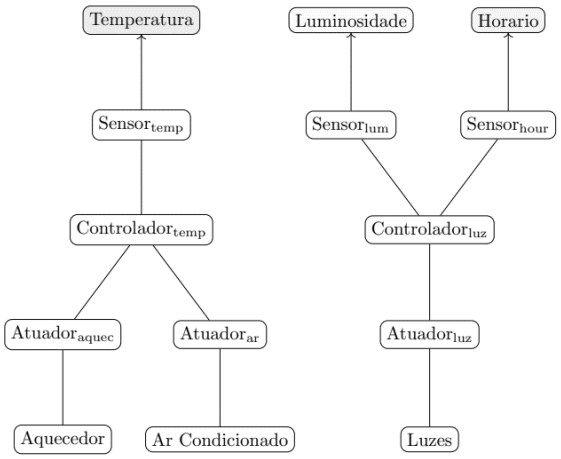

Figura 2. Diagrama de componentes

5. Especificação no Maude

O ambiente é formado por variáveis de ambiente (temperatura, luminosidade, horário) e componentes (sensores, controladores e atuadores). Os componentes são conectados por canais e escutam por eventos, ações e comandos transmitidos por estes. Estes componentes foram modelados utilizando uma sintaxe definida no Maude. A comunicação entre componentes foram modeladas utilizando regras de reescrita.

Devido a forma que o Maude faz as suas buscas (busca em largura), foi necessário realizar algumas alterações na nossa especificação. Se for permitido que diversas regras de reescrita ocorram a todo momento, o grafo de busca irá crescer de forma exponencial. Não haviam restrições na aplicação das regras de reescrita no modelo original, frequentemente gerando cenários irrelevantes para a evolução da rede. Por exemplo, a reemissão da mesma ação por dispositivos, mesmo que representando um comportamento real do sistema, induz a uma explosão no espaço de busca e que não permite a análise dos estados relevantes para a detecção de conflitos. Outro exemplo, quando a configuração permitia

que um dispositivo emitisse 2 mensagens (eventos, ações ou comandos) num mesmo instante para outro dispositivo, o Maude exploraria o cenário de reemissão alternada destas mensagens, causando um crescimento exponencial do espaço de busca.

Para eliminar estes cenários e diminuir o tamanho da árvore de busca, foram feitas as seguintes modificações:

- A comunicação entre componentes foi simplificada. Ao invés de modelar a comunicação entre cada componente como uma regra de reescrita, a cadeia de comunicação inteira foi modelada como uma única regra.
- Adição de logs para evitar que uma mesma comunicação ocorra repetidamente.
- O tempo de simulação da rede foi limitado a 5 dias. Sem esta restrição, se houver conflitos na configuração da rede, o Maude eventualmente irá encontrá-lo, mas se não houver conflitos, a busca continuaria indefinidamente.

Uma regra de reescrita condicional no Maude pode ser especificada utilizando a sintaxe:
`crl [nome-regra] : t1 => t2 if condição .`
Se a rede em algum momento atingir o estado $\mathbf{t_1}$ e a **condição** for verdadeira, a rede evolui para o estado $\mathbf{t_2}$.

Veja um exemplo simplificado de uma regra de reescrita utilizado no nosso trabalho:

```
crl [evento-temperatura-quente]
<TIMESTAMP = X>
<TEMPERATURA = Y>
=>
<EMITIR EVENTO> <EMITIR AÇÃO> <LIGAR AR CONDICIONADO>
<REGISTRAR LOG>
if Y == 25 and <EVENTO "Temperatura quente" AT X> not in
<LOGS> .
```

Devido ao tamanho e complexidade das nossas regras de reescrita (cada regra de reescrita tem cerca de 20 linhas), o exemplo acima foi simplificado para conter apenas as informações necessárias. Para ver a regra completa, veja o apêndice B. Quando a temperatura do quarto atinge 25ºC e o evento "**Temperatura Quente**" ainda não ocorreu, então o sensor de temperatura emite um evento que faz com que o controlador de temperatura emita uma ação que faz com que o atuador do ar condicionado emita um comando que liga o ar condicionado, e um log desde evento é registrado com um *timestamp*.

6. Resultados

Para validarmos o nosso modelo, foram instanciadas diversas configurações de componentes na rede, que foram simuladas no Maude à procura de conflitos.

A primeira configuração é a configuração completa especificada no capítulo 4, o Maude reporta corretamente que não existem conflitos nesta configuração. Esta configuração originalmente continha um conflito não percebido de antemão: originalmente as luzes foram configuradas para serem desligadas exatamente à meia noite, mas também há a regra de ambiente que diz que as luzes devem estar desligadas durante a madrugada. Quando o horário chegava à meia noite com as luzes ligadas, o Maude detectava

2 possíveis cenários para esta situação: “Está de madrugada e as luzes ainda estão ligadas, portanto temos um conflito direto” e “Desligar as luzes e prosseguir com a busca”. A configuração foi ajustada para que as luzes desliguem um pouco antes da meia noite, e o conflito foi resolvido.

Além da configuração principal, outras configurações foram testadas para validar o modelo em situações onde conflitos são esperados, que foram reportados corretamente pelo Maude. Para mais detalhes, algumas das configurações notáveis podem ser encontradas no endereço:
`https://github.com/mathamp/conflict-detection-maude`.

7. Conclusões

Apresentamos um modelo para detecção de conflitos e realizamos a sua especificação no Maude, que reportou corretamente os conflitos nos casos esperados e até mesmo em casos inesperados.

A especificação da concorrência deste sistema com regras de reescrita foi bastante natural, sendo considerado um aspecto positivo. Entretanto, como a configuração de cada componente é modelada como uma regra de reescrita, para testar novas configurações, é necessário escrever novas regras, que devido ao seu tamanho e complexidade, compromete a escalabilidade da abordagem e acaba dificultando o processo iterativo de novas configurações.

Certos aspectos da especificação do modelo tiveram que ser alterados para que o Maude possa fazer buscas em um tempo razoável. Essas alterações tem como principal objetivo reduzir o tamanho do espaço de busca.

Como sugestões de trabalhos futuros, os seguintes aspectos desse trabalho podem ser explorados:

- Expandir o domínio de estados dos objetos IoT de valores binários para valores discretos. Isto permitiria um controle mais refinado dos objetos, como controlar a temperatura do ar condicionado, ou a luminosidade das lâmpadas.
- Explorar a possibilidade de utilizar mais condições relacionais. Atualmente os eventos são disparados apenas se uma variável de ambiente atinge um valor exato.
- Este trabalho detecta somente conflitos diretos. Na literatura existem outros tipos de conflitos: indiretos e implícitos. Conflitos indiretos são possíveis de serem detectados formalmente, mas estes devem ser identificados de antemão, utilizando ferramentas de análise estatísticas. Por exemplo, ligar o ar condicionado e abrir a janela não são ações contraditórias, mas podem ser detrimentais para o ambiente. Para mais informações, veja [Adamczyk and Kliks 2023].
- Este trabalho aborda somente a detecção de conflitos. Mitigação automática de conflitos é uma possível extensão do trabalho que pode ser explorada.
- Identificar se existem regras de reescrita que podem ser trocadas por equações. Isto reduziria o crescimento do grafo de busca e permitiria buscas em configurações mais complexas.
- No Maude, é possível definir estratégias para aplicação de regras, que impediria a exploração de estados indesejáveis. O uso de estratégias pode ser usado para minimizar o crescimento da árvore de busca. [Durán et al. 2020]

Referências

Adamczyk, C. and Kliks, A. (2023). Conflict mitigation framework and conflict detection in o-ran near-rt ric. *IEEE Communications Magazine*, 61(12):199–205.

Al Farooq, A., Al-Shaer, E., Moyer, T., and Kant, K. (2019). Iotc2: A formal method approach for detecting conflicts in large scale iot systems. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 442–447.

Durán, F., Eker, S., Escobar, S., Martí-Oliet, N., Meseguer, J., Rubio, R., and Talcott, C. (2020). Programming and symbolic computation in maude. *Journal of Logical and Algebraic Methods in Programming*, 110:100497.

Lee, S. K., Bae, M., and Kim, H. (2017). Future of iot networks: A survey. *Applied Sciences*, 7(10).

Meseguer, J. (2012). Twenty years of rewriting logic. *The Journal of Logic and Algebraic Programming*, 81(7):721–781. Rewriting Logic and its Applications.

Pradeep, P. and Kant, K. (2022). Conflict detection and resolution in iot systems: A survey. *IoT*, 3(1):191–218.

A. Apêndice – Especificação completa da rede

Objetivos da rede:

- A temperatura do quarto deve estar entre 12ºC e 28ºC
- A luminosidade do quarto não deve estar abaixo de 500 lux
- Durante a madrugada, as luzes do quarto devem estar apagadas

3 sensores:

- Sensor de temperatura
- Sensor de luminosidade
- Relógio

2 controladores:

- Controlador para temperatura
- Controlador para lâmpadas

3 atuadores:

- Atuador para o ar condicionado
- Atuador para o aquecedor
- Atuador para lâmpadas

As variáveis de ambiente mudam conforme as seguintes regras:

- Temperatura (T):
  - A temperatura fora do quarto é dada por $T(h) = 10\sin(\frac{h-6}{12}\pi) + 20$
  - Se o ar condicionado está ligado, a temperatura do quarto cai 1ºC por hora
  - Se o aquecedor está ligado, a temperatura do quarto aumenta 1ºC por hora
  - Se nenhum dos dois está ligado, a temperatura do quarto se aproxima da temperatura de fora, 1ºC por hora
- Luminosidade (L):
  - A luminosidade do quarto é dada por $L(h) = 750\sin(\frac{h-6}{12}\pi) + 750$
  - Se as luzes do quarto estão ligadas, então a luminosidade do quarto é 1200 lux

Os dispositivos foram configurados em conjunto com as seguintes regras em mente:

- Se a temperatura do quarto atingir 25ºC, ligar o ar condicionado
- Se a temperatura do quarto atingir 20ºC, desligar o ar condicionado e o aquecedor
- Se a temperatura do quarto atingir 15ºC, ligar o aquecedor

- Se a luminosidade do quarto atingir 700 lux, ligar as luzes
- Se a luminosidade do quarto atingir 900 lux, desligar as luzes

- Quando o relógio atingir 23:00, forçar as luzes a ficarem desligadas
- Quando o relógio atingir 07:00, permitir que as luzes possam ser ligadas

B. Apêndice – Regra de reescrita completa para o evento de temperatura quente

```
crl [temp-hot-event] :
  Vars {
    AV "Time" is N1,
    AV "Temperature" is I1,
    CV "Air Conditioning" is St1,
    VarSet
  }
  Devs {
    Sensor "Temp" | event: E1,
      channels: (Ch1, ChSet1),
    Controller "Temp" | action: A1,
      channels: (Ch1, Ch2, ChSet2),
    Actuator "Air Conditioning" | command: C1,
      channels: (Ch2, ChSet3)
    DevSet
  }
  Logs {
    LogSet
  }
  =>
  Vars {
    AV "Time" is N1,
    AV "Temperature" is I1,
    CV "Air Conditioning" is on,
    VarSet
  }
  Devs {
    Sensor "Temp" | event: "TempHot",
      channels: (Ch1, ChSet1),
    Controller "Temp" | action: "Air Conditioning" On,
      channels: (Ch1, Ch2, ChSet2),
    Actuator "Air Conditioning" | command: on,
      channels: (Ch2, ChSet3)
    DevSet
  }
  Logs {
    Event "TempHot" at N1,
    Action "Air Conditioning" On at N1,
    Command "Air Conditioning" on at N1,
    LogSet
  }
  if (25 == I1) and
    not ((Event "TempHot" at N1) in LogSet) .
```