UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE SISTEMAS ELETRÔNICOS

MARIANA ARAÚJO TAVARES SATHLER

TRANSFORMAÇÃO DE REQUISITOS ESCRITOS EM PORTUGUÊS PARA PROPRIEDADES LÓGICAS TEMPORAIS

DISSERTAÇÃO DE MESTRADO

Joinville
2021

Mariana Araújo Tavares Sathler

TRANSFORMAÇÃO DE REQUISITOS ESCRITOS EM PORTUGUÊS PARA PROPRIEDADES LÓGICAS TEMPORAIS

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Sistemas Eletrônicos da Universidade Federal de Santa Catarina para a obtenção do título de mestre em Engenharia de Sistemas Eletrônicos.
Orientador: Gian Ricardo Berkenbrock

Joinville
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Sathler, Mariana Araújo Tavares
Transformação de requisitos escritos em português para propriedades lógicas temporais / Mariana Araújo Tavares Sathler ; orientador, Gian Ricardo Berkenbrock, 2021.
126 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Campus Joinville, Programa de Pós-Graduação em Engenharia de Sistemas Eletrônicos, Joinville, 2021.

Inclui referências.

1. Engenharia de Sistemas Eletrônicos. 2. Verificação formal. 3. Geração de propriedades temporais. 4. Redes neurais: aprendizado profundo. 5. Desenvolvimento dirigido a modelos. I. Berkenbrock, Gian Ricardo. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Sistemas Eletrônicos. III. Título.

Mariana Araújo Tavares Sathler

Transformação de requisitos escritos em português para propriedades lógicas temporais

O presente trabalho em nível de Mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Gian Ricardo Berkenbrock, Dr.
UFSC

Pablo Andretta Jaskowiak, Dr.
UFSC

Marco Aurélio Wehrmeister, Dr.
UTFPR

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Engenharia de Sistemas Eletrônicos.

Documento assinado digitalmente
Moises Ferber de Vieira Lessa
Data: 29/10/2021 18:39:02-0300
CPF: 062.157.286-16
Verifique as assinaturas em https://v.ufsc.br

Moisés Ferber de Vieira Lessa
Coordenador do Programa

Documento assinado digitalmente
Gian Ricardo Berkenbrock
Data: 29/10/2021 09:49:51-0300
CPF: 028.641.149-09
Verifique as assinaturas em https://v.ufsc.br

Gian Ricardo Berkenbrock
Orientador

Joinville, 27 de agosto de 2021.

RESUMO

Devido ao aumento na complexidade dos *softwares*, é fundamental encontrar meios que garantam que os *softwares* sejam seguros e livres de falhas. A verificação de modelos formais permite que falhas sejam encontradas ainda no início do projeto. A modelagem formal, no entanto, não é tão difundida na indústria quanto a modelagem semi-formal (como a UML). Este trabalho apresenta uma tradução de requisitos escritos em linguagem natural (Português) para propriedades temporais formais. As propriedades foram traduzidas utilizando duas abordagens de processamento de linguagem natural: tradução baseada em regras e tradução baseada em redes neurais. A partir da utilização dessas duas abordagens, as propriedades escritas em linguagem natural foram traduzidas para linguagem de entrada do verificador de modelos UPPAAL. O processamento de linguagem natural foi aplicado nos casos de estudo: Jantar dos Filósofos, Protocolo de Fischer e Base de Dados de Dwyer, para verificar a viabilidade do uso de um tradutor de propriedades. Os resultados obtidos foram satisfatórios.

Palavras-chave: Verificador de Modelos. UPPAAL. Geração de Propriedades Temporais, Aprendizado Profundo, Análise Léxica, Tradução baseada em regras, Tradução baseada em redes neurais.

ABSTRACT

The complexity of software has increased, and therefore it is essential to find ways to ensure that software is safe, and error free. The verification of formal models allows to find inconsistencies at the beginning of the project. Formal modeling, however, is not as used in industry as semi-formal modeling (such as UML). This work presents a translation of natural language writings - Portuguese for formal properties. The properties were formalized using two natural language processing approaches: Rule-based translation and neural network-based translation. From the use of these two approaches, properties written in natural language were translated to the formal input language of the UPPAAL verifier. To verify the possibility of using a property translator, natural language processing was found in the use cases: Philosophers' Dinner, Fischer Protocol and Dwyer Database. The results obtained were satisfactory.

Keywords: Model Checking. UPPAAL. Temporal Properties Translation, Deep Learning, Lexical Analysis, Rule-based translation, Neural network-based translation.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

As linguagens de projeto visual são uma forma de auxiliar e promover uma melhor captura de requisitos, além de facilitar o processo de arquitetura de *software*. Além disso, a linguagem de projeto visual permite compreender o comportamento do sistema antes que ele seja de fato, implementado (BOOCH; RUMBAUGH; JACOBSON, 2012).

No entanto, o uso de metodologias visuais não garante a correta construção do sistema, nem sua confiabilidade. A garantia da correção qualitativa de um sistema exige a verificação da integridade e consistência da especificação do sistema, e de atributos como o impasse (*deadlock*) e requisitos de segurança.

Uma maneira de aumentar a confiabilidade dos sistemas é através da utilização de métodos formais pelo uso de linguagens, técnicas e ferramentas matemáticas aplicadas à especificação, desenvolvimento e verificação de tais sistemas (CLARKE, 2004).

Wang (2004) afirma que a verificação formal significa explorar rigorosamente a correção dos projetos de sistema expressados como modelos matemáticos. Os métodos formais são técnicas capazes de reduzir a ambiguidade e inconsistência do projeto, minimizando o risco de falhas do software, onde uma falha de *software* significa a incapacidade de um produto executar uma função exigida ou a incapacidade de executar uma função entre os limites especificados (IEEE, 2010).

Por meio da utilização dos métodos formais é possível aplicar modelos e linguagem matemática nas etapas de construção do *software*, inclusive nas etapas de verificação e validação.

A etapa de verificação refere-se aos processos e técnicas usadas para assegurar que o modelo esteja correto e corresponda a quaisquer especificações e suposições acordadas, ou seja, certificar que o sistema atenda aos requisitos funcionais e não funcionais (BOURQUE; RICHARD, 2014).

Já a etapa de validação, refere-se aos processos e técnicas utilizadas para revisar e avaliar como o modelo funciona. Essa etapa serve para certificar-se que o sistema atenda as necessidades e expectativas (BOURQUE; RICHARD, 2014).

Apesar dos benefícios relacionados aos métodos formais, esses métodos não são amplamente utilizados na indústria devido principalmente, ao alto nível de abstração da modelagem matemática e as técnicas de análise (WANG, 2004).

Algumas das justificativas para não se utilizar métodos formais na indústria são a falta de compreensão, além da dificuldade de integração dos ciclos de desenvolvimento existentes e ausência de exigência explícita do mercado (LECOMTE et al., 2017).

1.1 JUSTIFICATIVA

A complexidade dos *sistemas de software* tem aumentado consideravelmente nos últimos anos (TAURION, 2005). No entanto, a atenção devotada aos projetos desses *sistemas de software* nas etapas de verificação, não tem avançado na mesma proporção. Um estudo feito por Dulac et al. (2005) mostra que o *software* teve sua participação nas causas de um conjunto de acidentes espaciais ocasionando a perda da missão.

Outro exemplo de falha aconteceu em uma máquina de radioterapia Therac-25 controlada por computador, deixando duas pessoas mortas e outras seis pessoas em sobredose por radiação (LEVESON; TURNER, 1993). Em 2009 e 2010 clientes da Toyota experimentaram uma aceleração não intencional em seus veículos o que levou a montadora a realizar milhões de *recalls* (LIKER; OGDEN, 2012).

Na *Aerospace Safety Advisory Panel* da NASA, foi abordado um problema de *software* identificado durante os testes de solo da *Starliner*. Se o problema não tivesse sido corrigido a tempo, teria interferido na separação do módulo de serviço e da cápsula *Starliner* o que seria uma falha catastrófica (LEWIS, 2020).

Estes exemplos mostram que a necessidade de implementar sistemas confiáveis e melhorar a qualidade do *software* tem se tornado cada vez mais importante. Visto que a complexidade do *software* tem aumentado, além da crescente dependência existente com o hardware.

Nesse cenário, a execução das etapas de verificação e validação podem representar um importante passo para a melhoria da qualidade dos sistemas de software. A verificação formal pode ser usada para reduzir o crescimento dos custos de verificação e integração e melhorar a qualidade dos projetos de sistemas na indústria (WANG, 2004).

No entanto, normalmente durante o desenvolvimento de um *software*, a forma comum de representar os requisitos é a abordagem escrita em linguagem natural, por isso esses requisitos não podem ser reconhecidos pelo verificador de modelos.

Gerar propriedades lógicas temporais não é uma tarefa intuitiva e o desenvolvedor precisaria aprender uma nova linguagem, além dos conceitos necessários para aplicação da verificação formal.

Um problema reconhecido na aplicação industrial é tornar o formalismo dessas propriedades mais intuitivo e fácil de usar.

1.2 PROBLEMA

A seguinte questão constitui o problema desta pesquisa: como obter propriedades lógicas temporais a partir dos requisitos do sistema escritos em português do Brasil para eles serem reconhecidos por um verificador de modelos?

1.3 METAS E OBJETIVOS DA PESQUISA

O ponto-chave do trabalho é produzir uma abordagem capaz de traduzir requisitos de sincronização e determinismo para uma notação formal capaz de ser reconhecida por um verificador de modelos.

Mais especificamente objetivam-se:

- Estudar os conceitos de verificação de modelo (*Model Checking*) e as linguagens formais de modelagem mais utilizadas para realização da verificação, visando definir uma linguagem que suporte a modelagem de sistemas concorrentes.
- Implementar um tradutor, capaz de traduzir requisitos do modelo, em propriedades formais temporais que possam ser verificadas.
- Aplicar a abordagem definida em um caso de estudo; e
- Analisar os resultados encontrados.

1.4 ESTRUTURA

Este trabalho está estruturado em cinco capítulos:

a) As bases teóricas são apresentadas no Capítulo 2, que inclui marcos conceituais importantes para o contexto de verificação formal e modelagem de sistemas, além de estudos sobre lógicas temporais e estado da arte sobre a tradução de requisitos em linguagem natural para propriedade formal.

b) A explanação sobre a abordagem do trabalho é descrita no Capítulo 3.

c) Os experimentos e resultados são analisados no Capítulo 4.

d) Finalmente, as considerações finais, limitações e trabalhos futuros são apresentados no Capítulo 5.

2 REFERENCIAL TEÓRICO

Nesse capítulo são revisados os conceitos relacionados a esse trabalho. A revisão segue a seguinte ordem: aspectos para modelagem semi-formal e depois a modelagem formal. Em seguida, uma breve revisão de verificação de modelos é realizada e, assim, segue para técnicas de tradução de requisitos em linguagem natural para propriedades formais. Concluindo então com a descrição de técnicas de processamento de linguagem natural e os trabalhos relacionados.

2.1 MODELAGEM SEMI-FORMAL DE SISTEMAS

A modelagem é uma das principais atividades que levam à implementação de um bom sistema, e de acordo com Booch, Rumbaugh e Jacobson (2012), há quatro objetivos principais para se criar modelos:

1. Eles auxiliam a visualização de como é o sistema ou como se deseja que ele venha ser construído;
2. Permitem especificar a estrutura ou o comportamento de um sistema;
3. Proporcionam um guia para a construção do sistema;
4. Por fim, eles documentam as decisões tomadas no projeto.

A modelagem implica na construção de modelos do sistema, onde é possível visualizar seus componentes, o relacionamento entre eles e como se comportam. Ela permite que os projetistas compreendam melhor o sistema, possibilitando que erros sejam eliminados ainda no projeto de *software*, colaborando para a obtenção de um produto mais robusto e seguro.

Uma das linguagens de modelagem que se difundiu no meio acadêmico e profissional, foi a *Unified Modeling Language (UML)*. Existem estudos como o de Petre (2013) o qual aponta que, na prática a UML não é utilizada com abrangência universal. Foi concluído que a maioria dos 50 entrevistados nesse estudo, não utilizavam a UML e os que utilizavam, realizam essa prática informalmente.

No entanto, o estudo Störrle (2017) apresenta uma conclusão contrária a conclusão reportada por Petre (2013). Em Störrle (2017) foram entrevistados 96 participantes e apesar dos autores concordarem que o uso informal da UML é mais comum do que o uso formal, observou-se que a UML é amplamente utilizada. Considerando que o seu uso formal não seja universal, há uma população considerável que usa a UML na indústria.

A diferença nas duas constatações é que Petre (2013) se concentrou apenas nos usos muito formais da UML, como a geração de código, e a pesquisa realizada em Störrle (2017) confirma ser o cenário menos comum para a utilização da UML.

Outra diferença é a região em que foram feitas as pesquisas, Petre (2013) se concentrou na América do Norte e no Reino Unido, já no estudo de Störrle (2017) foram utilizadas respostas de um número considerável de participantes de outras regiões como Alemanha, Índia, Escandinávia, ou seja, é possível que as diferenças regionais e até mesmo culturais sejam alguns dos motivos para as diferentes conclusões.

Considerando esse cenário a UML será apresentada na seção seguinte.

2.1.1 Unified Modeling Language - UML

A UML foi proposta por Grady Booch, James Rumbaugh e Ivar Jacobson e pode ser descrita como uma linguagem de modelagem visual de propósito geral. Ela serve para visualizar, especificar, construir e documentar o sistema de *software*. Esta linguagem é uma maneira de padronizar a modelagem orientada a objetos de forma que qualquer sistema possa ser modelado da maneira apropriada, simples de ser atualizado e compreendido (BOOCH; RUMBAUGH; JACOBSON, 2012).

A UML oferece diagramas para se modelar sistemas em duas macro-visões diferentes: uma estrutural e outra comportamental (BOOCH; RUMBAUGH; JACOBSON, 2012). A primeira representa o *software* na sua forma estática, descrevendo sua arquitetura. Por outro lado, a modelagem comportamental tem por objetivo representar uma visão dinâmica do *software* com eventos disparados, ações executadas, trocas de mensagens, e transição de estados. Assim, conforme as macro-visões, a UML propôs os seguintes diagramas divididos entre estruturais e comportamentais:

a) Diagramas Estruturais: Os diagramas estruturais mostram a estrutura estática do sistema e suas partes em diferentes níveis de abstração e como elas se relacionam.

b) Diagramas Comportamentais: Os diagramas comportamentais mostram a natureza dinâmica dos objetos do sistema, que pode ser descrita como uma série de mudanças no sistema com o passar do tempo (BOOCH; RUMBAUGH; JACOBSON, 2012).

Entre os diagramas estruturais está o diagrama de classes. Esse diagrama permite visualizar as classes que compõem o sistema. Além disso, descreve os relacionamentos estáticos entre elas. Essa categoria de diagrama possui componentes importantes para sua construção:

a) Propriedades(Atributos);

b) Operações(métodos);

c) Relacionamentos e Restrições.

Entre os diagramas comportamentais está o diagrama de estados. Esse diagrama mostra os vários estados possíveis por quais um objeto pode passar. Um objeto muda de estado quando acontece algum evento interno ou externo ao sistema. Atra-

vés da análise das transições entre os estados, pode-se prever todas as possíveis operações realizadas, em função de eventos que podem ocorrer.

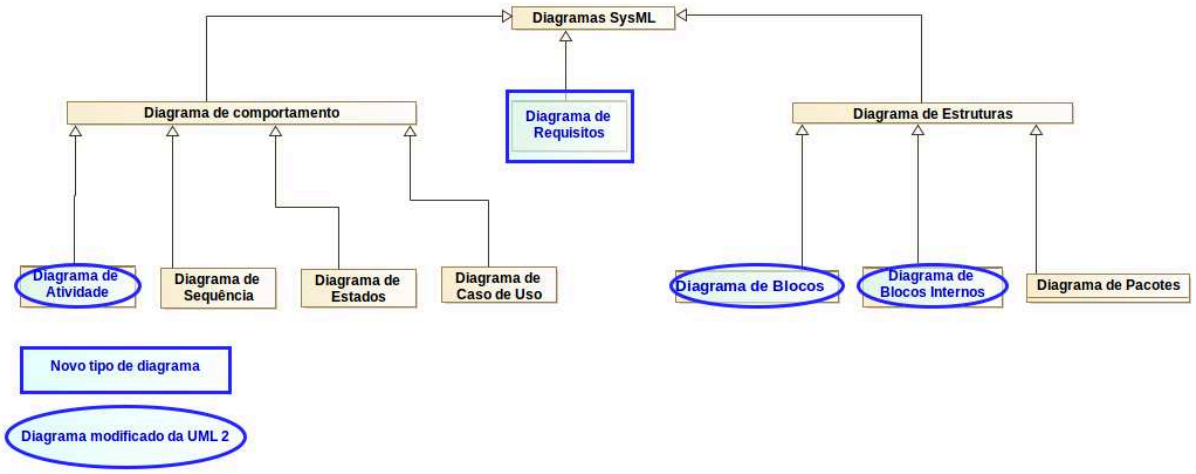
Como a UML é uma especificação genérica, alguns recursos de domínios específicos não são suportados (BOOCH; RUMBAUGH; JACOBSON, 2012). No entanto a UML permite a utilização de extensões que possibilita uma melhor expressividade na modelagem de sistemas (GHERBI; KHENDEK, 2006). Uma dessas extensões é a *Systems Modeling Language* (SysML) que foi originalmente desenvolvida em um projeto de código aberto pela *Object Management Group* (OMG) em conjunto com o Conselho Internacional de Engenharia de Sistemas (INCOSE)(LENNY, 2014).

Adicionalmente, a SysML tem como propósito geral especificar, analisar, projetar e verificar sistemas complexos que podem incluir hardware, software, procedimentos e instalações.

A SysML é definida como uma extensão de um subconjunto da UML usando o mecanismo de perfis existente na UML. O suporte para representar requisitos e vincular esses requisitos ao modelo do sistema, foi um dos recursos adicionados pela SysML. A linguagem fornece representações gráficas com uma base semântica para modelar requisitos (OMG, 2007).

A Figura 2.1 apresenta os diagramas contidos no SysML. Além da SysML adicionar o diagrama de requisitos, o diagrama de estados foi mantido pela SysML, já o diagrama de classes foi substituído pelo diagrama de blocos.

Figura 2.1 – Diagramas SysML



Fonte: Adaptado de OMG (2007).

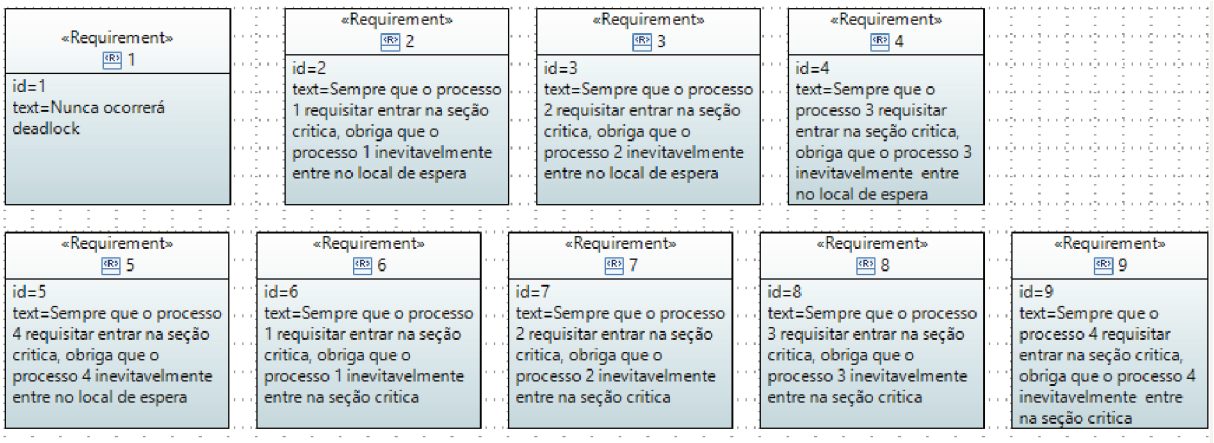
Os diagramas de requisitos, bloco e estado serão descritos nas próximas seções.

2.1.2 Diagrama de requisitos

A SysML fornece um diagrama para modelagem de requisitos, onde um requisito é representado pelo estereótipo “Requirement”. Entre os atributos do diagrama de requisitos, estão *id* e *text*, onde são definidos um identificador único e o conteúdo do requisito respectivamente (LENNY, 2014). Esses atributos podem ser observados na Figura 2.2.

Propriedades adicionais, como *status* da verificação e prioridade também podem ser especificados. O diagrama de requisitos pode representar os requisitos em formato gráfico, tabular ou de estrutura em árvore. Um requisito também pode aparecer em outros diagramas para mostrar seu relacionamento com outros elementos de modelagem.

Figura 2.2 – Diagrama de Requisitos - Protocolo de Fischer



Fonte: Elaborado pelo Autor

2.1.3 Diagrama de blocos

O diagrama de blocos (BDD) é utilizado para descrever a estrutura de um sistema. Essa descrição é feita a partir da utilização de propriedades, operações e relacionamentos. O diagrama de blocos estende o diagrama de classes da UML. O diagrama de blocos é separado em setores, onde é possível definir o nome do bloco, precedido do esteriótipo “Block”. Entre os demais setores estão os que definem as propriedades opcionais do bloco como: Valores (*Values*) ou Atributos (*attributes*) e Operações (*Operations*) descritos abaixo (LENNY, 2014):

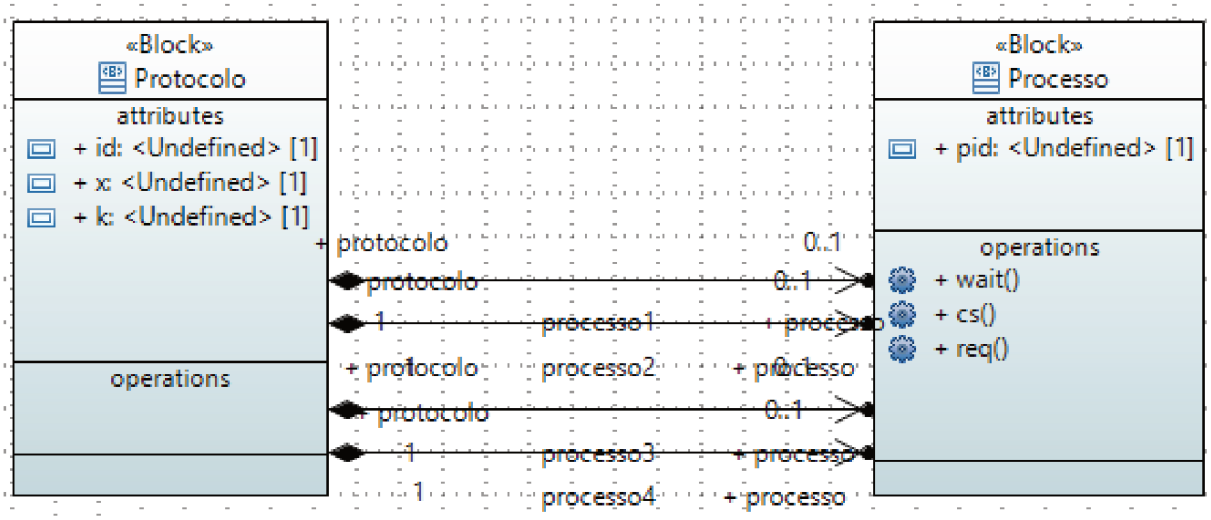
a) Atributos: Essa propriedade é definida no setor de Atributos de um bloco, e pode representar uma quantidade de um tipo específico, como um *booleano* ou uma *string*, por exemplo.

b) Operações: Uma operação representa um comportamento que um bloco executa quando é chamado. Uma operação é invocada por um evento de

chamada. O nome da operação é definida pelo responsável pela modelagem, além disso, uma operação pode conter parâmetros, e também um retorno.

A Figura 2.3 exemplifica a estrutura do diagrama de blocos:

Figura 2.3 – Diagrama de Blocos - Protocolo de Fischer



Fonte: Elaborado pelo Autor.

2.1.4 Diagrama de estados

Os diagramas de estados, ou maquina de estados pode ser utilizado para expressar o comportamento dinâmico de um sistema. O diagrama de estados consegue mostrar como uma estrutura dentro de um sistema muda de estado em resposta a ocorrências de eventos ao longo do tempo (LENNY, 2014).

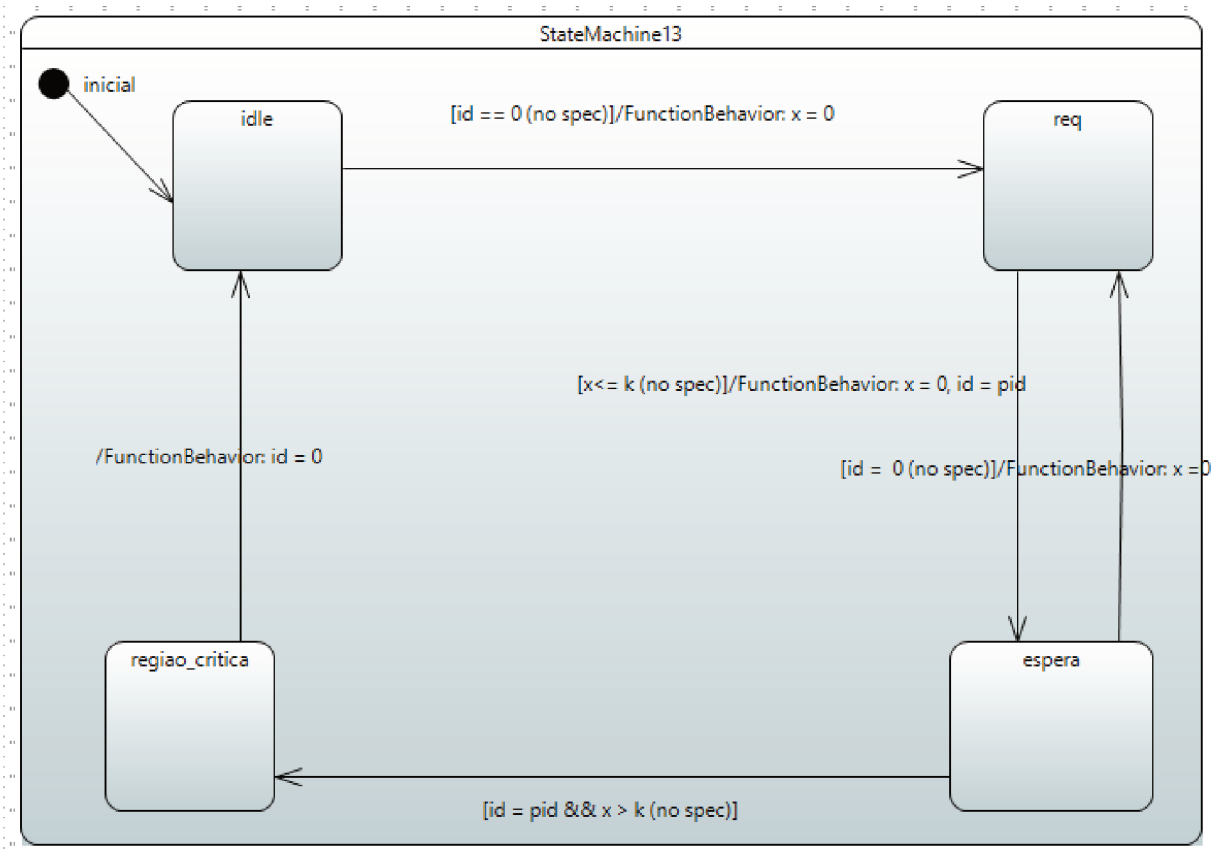
a) Estados: Um arquivo, por exemplo, pode possuir os seguintes estados: $aberto$, $fechado$, $modificado$, $não\ modificado$, $criptografado$, entre outros. Alguns estados só fazem sentido no contexto de outros estados. Dizer que o arquivo está no estado modificado ou não modificado, só faz sentido se esse arquivo estiver no estado $aberto$, nesse caso o estado $aberto$ é definido como um estado composto, e os estados $modificado$ e $não\ modificado$ são sub-estados do estado $aberto$. Um estado que não possui sub-estados é chamado estado simples. Além do estado composto e estado simples, também é comum encontrar um estado final.

b) Transição: Uma transição representa a mudança de um estado para outro, ou uma mudança de um estado de volta para ele mesmo. A transição é representada por uma linha sólida com uma ponta de seta aberta desenhada de um vértice de origem para um vértice de destino. Os vértices de origem e destino normalmente são os estados. Cada transição pode especificar três informações opcionais: um gatilho, uma guarda e

um efeito. Essas informações aparecem nas transições da seguinte forma: $< gatilho > [< proteção >]/ < efeito >$.

A Figura 4.6 exemplifica o diagrama de estados:

Figura 2.4 – Diagrama de estados - Protocolo de Fischer



Fonte: Elaborado pelo Autor.

Como descrito em seções anteriores, a UML consegue representar um sistema através de várias perspectivas diferentes. Através da utilização de suas extensões (como o SysML). No entanto, não é possível a aplicação de verificação formal sobre esses modelos. Considerando esse cenário, surge a necessidade de modelar e especificar esses sistemas formalmente. Na próxima seção serão apresentados conceitos relacionados a modelagem formal.

2.2 MODELAGEM FORMAL DE SISTEMAS

Na visão de especificação e verificação, um modelo é um comportamento de uma descrição do sistema (ou especificação). Um modelo pode ser um conjunto de estados, uma sequência de estados, uma sequência de eventos, uma árvore de estados ou um domínio infinito com relações (WANG, 2004).

A modelagem formal consiste em construir um modelo formal do sistema, que deve ser aceito por uma ferramenta de verificação de modelos e a partir deste mo-

delo, obter todos os comportamentos possíveis do sistema. Dentre as abordagens de modelagem formal, existe o sistema de transição (ST).

Segundo Baier e Katoen (2008), os sistemas de transição são frequentemente usados na ciência da computação como modelos para descrever o comportamento dos sistemas. Eles são definidos como grafos direcionados onde os nós representam os estados e as arestas modelam as transições, ou seja, as mudanças de estado. Um estado descreve algumas informações sobre um sistema em um determinado momento de seu comportamento. Por exemplo, o estado de um semáforo indica a cor atual do semáforo.

No caso do semáforo, uma transição pode indicar uma mudança de uma cor para outra, enquanto para um programa sequencial uma transição normalmente corresponde à execução de uma declaração e pode envolver a alteração de algumas variáveis. Em um sistema de transição que modela uma aplicação sequencial, os estados terminais também ocorrem e representam o término do programa.

O sistema de transição (ST) para Baier e Katoen (2008) é definido por uma tupla *{ S, Act,* $\rightarrow$*, I, AP, L}*, onde:

- *S* É definido como o conjunto de estados.
- *Act* Conjunto de ações.
- $\rightarrow \subseteq$ *S x Act x S* Relação de transição.
- *I* $\subseteq$ *S* Conjunto de estados iniciais.
- *AP* Conjunto de proposições atômicas.
- *L:* $S \rightarrow 2^{AP}$ Função de rotulagem.

Segundo Baier e Katoen (2008), o sistema de transição começa em algum estado inicial s_0 e evolui conforme a relação de transição $\rightarrow$. Se s é o estado atual, então uma transição $s \rightarrow^a s^{'}$ originando de s é selecionada não deterministicamente e executada, ou seja, a ação a é realizada e o sistema de transição evolui do estado s para o estado $s^{'}$. Este procedimento de seleção é repetido no estado $s^{'}$ e termina quando um estado sem transições é encontrado.

Para exemplificar o ST, Baier e Katoen (2008) usam um modelo do comportamento de uma máquina de venda de bebidas. Na máquina, o cliente pode escolher refrigerante ($soda$) ou cerveja ($beer$). Basicamente, pay é o estado inicial, onde a máquina fica aguardando o pagamento, ou seja, a realização da ação $insertcoin$, após a ação do pagamento a máquina fica aguardando a seleção da bebida escolhida ($soda$ ou $beer$). Após a seleção da bebida, a máquina vai para o próximo estado de entregar a bebida selecionada. Na Figura 2.5 os estados são representados pelas Figuras circulares e as transações são as bordas com seus respectivos rótulos:

a) O conjunto de estados é dado por: $S = \{pay, select, soda, beer\}$.

b) O estado inicial é dado por $I = \{pay\}$

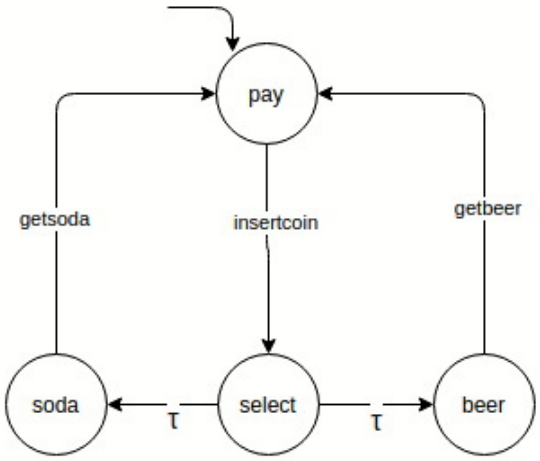
As ações permitidas na máquina são inserir moeda ($insertCoin$), pegar o refrigerante ($getSoda$) e pegar a cerveja ($getBeer$). O conjunto de ações é dado por:

$$Act = \{insertCoin, getSoda, getBeer, \tau\}$$

Abaixo, é demonstrado um exemplo de transição:

$$pay \xrightarrow{insertCoin} selectBeer \xrightarrow{getBeer} pay$$

Figura 2.5 – ST da máquina de venda de bebidas



Fonte: Adaptado de Baier e Katoen (2008, p. 21).

No modelo, as transições também podem expressar uma categoria de condição chamada guarda. Assumindo que os estados $start$, $select$ descrevem os estados da máquina de bebidas, após a inserção da moeda e seleção da bebida, a máquina deve verificar se existe a bebida selecionada: *select* $\xrightarrow{soda==0 \ and \ beer==0 \ : \ ret\_coin}$ *start*.

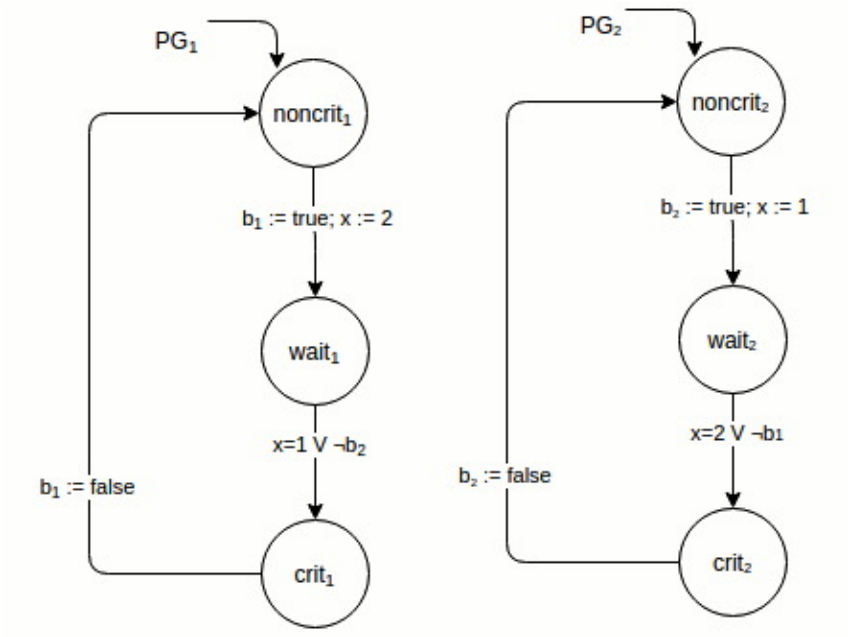
Nessa transição após a ação do $select$, a máquina verifica se ainda existem as bebidas no estoque, se não existir ($soda == 0 \, and \, beer == 0$), a máquina devolve a moeda (ret_coin) e volta para o início ($start$) aguardando nova ação.

Outro exemplo abordado por Baier e Katoen (2008) é a modelagem de dois processos que possuem regiões críticas onde recursos são compartilhados auxiliando na representação de uma modelagem em ST de sistemas concorrentes.

Na Figura 2.6 o modelo é composto pelos processos PG_i, onde $i = \{1, 2\}$, os processos são compostos pelas variáveis $Var = \{x, b_1, b_2\}$ com os estados $noncrit_i, wait_i, crit_i$. A variável x representa o índice do processo (1 ou 2). Um dos

mecanismos utilizados em modelos concorrentes, são as variáveis compartilhadas, onde os dois processos são capazes de “enxergar” essas variáveis. No entanto, esse tipo de tratativa não garante exclusão mútua, ou seja, não garante acesso exclusivo ao recurso compartilhado (BAIER; KATOEN, 2008).

Figura 2.6 – Algoritmo de Peterson

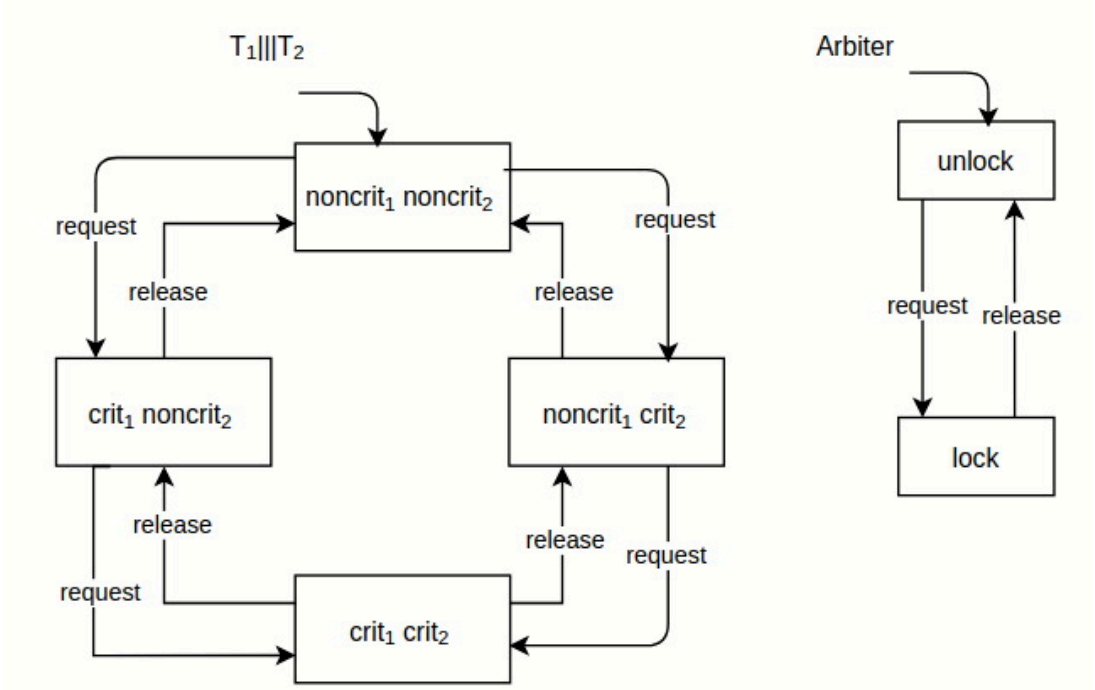


Fonte: Adaptado de Baier e Katoen (2008, p. 43).

Segundo Baier e Katoen (2008), para processos concorrentes existe o conceito de *“handshaking”* (aperto de mãos). O *“handshaking”* descreve que processos simultâneos que desejam interagir podem fazer isso de forma assíncrona. Os processos podem interagir apenas se ambos estiverem participando dessa interação simultaneamente, e então eles “apertam as mãos”.

Baier e Katoen (2008) fornecem uma solução para o exemplo da Figura 2.6, utilizando um semáforo representado por um processo chamado *Arbiter* demonstrado na Figura 2.7.

Figura 2.7 – Algoritmo de Peterson usando Handshaking



Fonte: Adaptado de Baier e Katoen (2008, p. 51).

Para simplificação, o modelo considerou somente as ações $noncrit_i, crit_i$. O *Arbiter* define se é possível que o processo entre na seção crítica. Ele permite dois estados $unlock, lock$ sendo que o estado $lock$ significa que um dos processos já está acessando a seção crítica e o outro deverá aguardar até que seja liberado e o *Arbiter* vá para o estado $unlock$. Observando o modelo é possível perceber que o estado $\{crit_1, crit2\}$ nunca será atingido visto que não é possível fazer requisições simultâneas, pois o recurso ainda estará com *lock* pelo *Arbiter*.

Para a modelagem de sincronização, é possível utilizar um recurso chamado canal nas transações. Os canais são como *buffers* de primeiro a entrar, primeiro a sair. A utilização desse recurso permite que os processos do sistema modelado possam comunicar entre si (BAIER; KATOEN, 2008):

Quando uma transação contendo como guarda: $chan!var1$ é acionada, significa que o valor de $var1$ está sendo transmitido pelo canal $chan$.

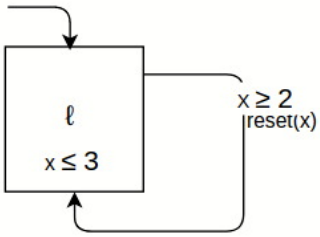
Se outro processo está aguardando o valor enviado por $chan$, após recebimento ele poderá ir para o próximo estado. Por exemplo: $chan?var2$ recebe a mensagem do canal $chan$ e atribui o valor a variável $var2$.

Existem ainda os autômatos temporizados que segundo Baier e Katoen (2008), são uma categoria de grafo que contém um conjunto finito de variáveis de relógio. A diferença entre as variáveis de relógio e as variáveis comuns é que as variáveis de relógio só podem ser zeradas ou monitoradas, diferente das variáveis comuns que podem ser alteradas por alguma transação. A variável de relógio simula a quantidade de tempo decorrido desde a última reinicialização dessa variável. Além disso, as variá-

veis de relógio podem ser inicializadas independentemente de outra variável de relógio. Condições que dependem dos valores de relógio são chamadas restrições de relógio (*clock*).

A Figura 2.8 mostra o exemplo de um autômato temporizado, onde foi adicionado uma invariante $x \leq 3$ e uma guarda $x \geq 2$, onde x é definido como uma variável de relógio (BAIER; KATOEN, 2008).

Figura 2.8 – Autômato temporizado



Fonte: Adaptado de Baier e Katoen (2008, p. 680).

No exemplo, o processo só pode permanecer no estado ℓ enquanto $x \leq 3$. Quando $x \geq 2$, o processo realiza uma transição (auto *loop*) onde o valor de x é reiniciado e volta a valer 0 novamente.

Para que modelos sejam verificados, eles devem ser acompanhados de uma especificação da propriedade que se deseja verificar. A seguir serão apresentados os conceitos relacionados a essas propriedades.

2.3 PROPRIEDADES DE TEMPO LINEAR

As propriedades de tempo linear especificam os traços que um sistema de transição deve exibir. Uma propriedade de tempo linear especifica o comportamento admissível (ou desejado) do sistema em consideração. Baier e Katoen (2008) apresentam os seguintes conceitos sobre os tipos de propriedades de tempo linear:

a) Segurança (*Safety*): as propriedades de segurança consideram que em um tempo finito, uma situação considerada indesejável não ocorra. Ou seja, nada de ruim deve acontecer. Por exemplo, a propriedade: “sempre, apenas um processo está em sua seção crítica” é uma propriedade de segurança.

   Para a máquina de venda de bebidas um exemplo de requisito de segurança seria: “O número de moedas inseridas é sempre igual ao número de bebidas dispensadas.”

   Esse requisito expressa a ideia de que a máquina de bebidas só deverá dispensar a bebida após uma moeda ser inserida como pagamento, ou seja, a máquina nunca dispensa uma bebida sem que o pagamento seja realizado.

Durante a verificação, os estados devem ser analisados assegurando que em nenhum caminho a propriedade de segurança seja violada.

b) Impasse (Deadlock): para sistemas concorrentes não é desejável que eles fiquem bloqueados. Nos processos exemplificados na Figura 2.6 sobre exclusão mútua, os estados terminais são indesejáveis, assim representam um erro na modelagem. Geralmente, tais estados terminais indicam a presença de um impasse (*deadlock*). Um cenário de impasse ocorre, por exemplo, quando os componentes aguardam mutuamente o progresso do outro.

c) Vivacidade (*Liveness*): as propriedades de vivacidade complementam as propriedades de segurança garantindo que o sistema funcione como o esperado no tempo infinito. Essa propriedade afirma que em algum momento o sistema irá realizar algo bom, ou seja, algum comportamento esperado irá ocorrer. Para o exemplo da Figura 2.6 exclusão mútua, a propriedade de segurança afirma que os processos nunca estão simultaneamente em suas seções críticas, já a propriedade de vivacidade afirmam que: “Eventualmente cada processo entrará em sua seção crítica”.

d) Imparcialidade (*Fairness*): as propriedades de imparcialidade são caracterizadas pelos caminhos onde as transições habilitadas são executadas de forma imparcial. Uma execução imparcial é caracterizada pelo fato de que certas restrições de justiça são atendidas. Para o exemplo da Figura 2.6 é possível expressar a seguinte propriedade: “Cada processo que solicita acesso à seção crítica eventualmente será capaz de acessá-la”.

2.4 LÓGICA TEMPORAL

O conceito de Lógica Temporal surgiu nos anos de 1950 sendo empregado principalmente por lógicos e cientistas da computação. A lógica temporal é utilizada para formalizar o entendimento e definir as semânticas de expressões temporais em linguagem natural (EMERSON, 1990).

Na lógica temporal, o tempo é visto como possíveis sequências de estados associados às suas transições. O estado é definido como uma descrição de um sistema em um determinado instante de tempo, e as transações são vistas como uma relação entre dois estados. As observações sobre o comportamento de um sistema em um determinado tempo, são feitas utilizando propriedades expressas como fórmulas de uma linguagem de lógica temporal (EMERSON, 1990).

Como característica principal, a lógica temporal apresenta que uma determinada fórmula lógica pode apresentar valores distintos em instantes diferentes do tempo. Existem dois tipos básicos de lógicas temporais: Linear Temporal Logic (LTL) e Computation Tree Logic (CTL).

Para facilitar o entendimento, duas definições são descritas a seguir:

a) Tempo Linear: É quando o comportamento do sistema pode ser representado por um conjunto de traços infinitos que começam no estado inicial.

b) Tempo Ramificado: O sistema é representado por uma árvore computacional, e possui uma profundidade ilimitada. O estado inicial representa a sua raiz.

2.4.1 Lógica Temporal Linear (LTL)

A lógica temporal linear (LTL) tem como base a lógica modal, apresentada no Apêndice B, e adiciona em sua representação a ideia de tempo. Essa lógica permite verificar se uma sentença é verdadeira ou não ao longo do tempo. Por exemplo, uma afirmação pode ser falsa agora, mas em algum momento poderá se tornar verdadeira (C. CUNHA G. B., 2017).

Uma fórmula LTL válida sintaticamente é formada pelas variáveis proposicionais, os conectivos da lógica proposicional ($\neg, \bigvee, \bigwedge, \rightarrow$), e os operadores temporais descritos na Tabela 2.1 (ROZIER, 2011):

Tabela 2.1 – Operadores Temporais LTL

X	$X\alpha$	α é verdadeiro no próximo estado
F	$F\alpha$	α é válido em algum estado do caminho
G	$G\alpha$	α é sempre válido
$\cup$	$\alpha \cup \beta$	α é verdadeiro no caminho até que β seja verdadeiro.

Fonte: Elaborado pelo Autor.

Já no contexto semântico, existem alguns padrões mais frequentemente aplicados:

- Ausência: Quando no contexto se pretende que não ocorra certos eventos ou estados.
- Universalidade: Quando se pretende que em todo o contexto, certa propriedade se verifique.
- Existência: Quando se pretende que uma propriedade ocorra alguma vez no contexto.
- Resposta: Dentro do contexto a ocorrência de certo evento (causa) deve ser seguida da ocorrência de outro evento (efeito).

A gramática LTL pode ser vista conforme (BAIER; KATOEN, 2008):

$$\varphi := \textit{true} | a | \neg\varphi | \varphi_1 \wedge \varphi_2 | \textit{operador}\varphi | \varphi_1 \cup \varphi_2$$

2.4.2 Lógica Computacional em Árvore (CTL)

A semântica da CTL não está baseada em uma noção linear de tempo, mas em uma noção ramificada de tempo, ou seja, uma árvore infinita de estados. O tempo de ramificação refere-se ao fato de que a cada momento pode haver vários futuros possíveis diferentes. Essa classe de lógica temporal é conhecida como lógica temporal ramificada, devido a essa noção de tempo ramificado. A semântica de uma lógica temporal ramificada é definida em termos de uma árvore infinita e dirigida de estados, em vez de uma sequência infinita. Cada transição da árvore representa um único caminho. A árvore em si, representa, todos os caminhos possíveis.

Para a CTL existem os quantificadores de caminho, onde (ROZIER, 2011):

- *A*: representa a ideia de "para todo caminho"
- *E*: representa a ideia de "existe um caminho"

Também, existem os operadores de tempo linear definidos na Tabela 2.2:

Tabela 2.2 – Operadores Temporais CTL

Xp	p é válida no próximo estado (*next*)
Pp	p foi válida em algum estado passado(*past*)
Fp	p é válida em algum estado futuro (*future*).
Hp	p foi válida em todos os estados passados.
Gp	p é válida globalmente no futuro (*globally*).
pUq	p é válida até que q seja válida (*until*).

Fonte: Elaborado pelo Autor.

A gramática CTL pode ser escrita da seguinte forma:

$$\varphi := \textit{true}|a|\neg\varphi|\varphi_1 \wedge \varphi_2|E\Phi|A\Phi$$

Onde:

$$\Phi := \textit{operador}\varphi|\varphi_1 \cup \varphi_2$$

onde

$$\varphi, \varphi_1, \varphi_2$$

são estados.

As lógicas de LTL e CTL podem especificar a ordem dos comportamentos de um sistema. Essas lógicas podem especificar a ordem dos eventos e ações, e também podem ser utilizadas para verificar as propriedades de acessibilidade, vivacidade e segurança.

As lógicas temporais fornecem um formalismo para o raciocínio qualitativo sobre a mudança ao longo do tempo. Por exemplo: Fp significa que "um evento p ocorre em algum momento no futuro." Essa afirmação não impõe limite ao tempo que pode decorrer antes da ocorrência de p.

A utilização da modelagem formal, juntamente com as propriedades lógicas possibilitam a utilização de técnicas de verificação formal. Os conceitos sobre o processo de verificação formal serão apresentados na próxima seção.

2.5 VERIFICAÇÃO DE MODELOS FORMAIS (MODEL CHECKING)

Técnicas formais são usadas para melhoria de processos de verificação de *software*. A verificação formal pode seguir duas linhas diferentes de raciocínio: A prova de teoremas e a verificação de modelos. Na prova de teoremas é utilizada uma técnica formal para confirmar se a implementação de um modelo está de acordo com a sua especificação. Já na verificação de modelos, são criadas máquinas de estados finitos para assegurar a validade de uma propriedade (SOBEIH; VISWANATHAN; HOU, 2004).

Durante o desenvolvimento de *software* e *hardware* de sistemas complexos, gasta-se mais tempo e esforço nas etapas de verificação do que propriamente na construção. Os métodos formais fornecem técnicas de verificação que podem reduzir o tempo gasto nas etapas de verificação (BAIER; KATOEN, 2008). Baier e Katoen (2008) afirmam que os métodos formais são altamente recomendados para o desenvolvimento de *software* de sistemas críticos segundo as práticas da International Electrotechnical Commission (IEC) e os padrões da European Space Agency (ESA).

Dentre as técnicas de verificação fornecidas pelos métodos formais, está a verificação de modelo (Model Checking). A técnica de verificação de modelo foi desenvolvida por volta dos anos 80 por Clarke e Emerson e por Queille e Sifakis (BAIER; KATOEN, 2008).

O método de Verificação *Model Checking* ou Verificação de Modelos, consiste na exploração de máquina de estados finitos, ou estados infinitos que podem ser reduzidos a finitos. Essa técnica baseia-se em modelos que descrevem o possível comportamento do sistema. A modelagem precisa dos sistemas pode levar a descoberta de problemas como ambiguidade e inconsistência nas especificações do sistema. Segundo Baier e Katoen (2008) a verificação de modelo pode ser comparada a um sistema computacional de xadrez que verifica possíveis movimentos.

A verificação de modelo examina todos os possíveis cenários do sistema de

forma sistemática. Se uma propriedade não for verdadeira, o verificador de modelo produz um contra-exemplo mostrando como a propriedade pode ser violada (MILLER; WHALEN; COFER, 2010).

Aplicar a verificação de modelos a um projeto consiste nas seguintes fases (BAIER; KATOEN, 2008):

a) Fase de modelagem: nessa fase é construído o modelo do sistema em consideração usando a linguagem de descrição do verificador de modelo que será utilizado. Como uma primeira verificação e avaliação rápida do modelo, devem ser executado algumas simulações e então deve ser formalizados as propriedades a serem verificadas usando o idioma da especificação de propriedade.

b) Fase de execução: a fase onde é executado o verificador do modelo para verificar a validade da propriedade no modelo do sistema.

c) Fase de análise: nessa fase acontece a atividade de análise dos resultados da verificação. Se a propriedade foi satisfeita, deve-se verificar a próxima propriedade (se houver). Se a Propriedade foi violada, deve-se:

   – Analisar o contraexemplo gerado por simulação;
   – Refinar o modelo, design ou propriedade;
   – Repetir todo o procedimento.

Mais detalhadamente, para a verificação de modelos é necessário fornecer um modelo de estados finitos de um sistema e uma propriedade a ser verificada, para então confirmar a validade dessa propriedade através da inspeção dos caminhos possíveis do espaço de estados. Durante a fase de execução, as propriedades em lógica temporal são executadas e analisadas pelo verificador.

A seguir, estão algumas (mas não todas) ferramentas proeminentes de verificação de modelos de domínio público, muitas das quais estão disponíveis gratuitamente para uso não comercial.

2.5.1 NuSMV

A ferramenta de verificação de modelos NuSMV foi escrita em ANSI C e é uma ferramenta para a verificação formal de sistemas de estados finitos. A ferramenta possui código aberto, o que permite que programadores contribuam desenvolvendo melhorias. Essa ferramenta foi originada da reengenharia, reimplementação e extensão do *Symbolic Model Verifier (SMV)*, o verificador de modelo baseado em *Binary Decision Diagrams (BDD)* (CIMATTI et al., 2002).

O NuSMV pode descrever máquinas de estados finitos, e pode utilizar os seguintes tipos de dados na linguagem: *booleanos*, escalares, vetores de bit e matrizes

fixas de tipos de dados básicos. Além disso, a ferramenta possui um *shell* de interação textual e interface gráfica e permite a verificação de lógicas LTL e CTL.

A linguagem SMV usa um algoritmo de busca baseado em *Binary Decision Diagrams (BDD)* que determina se o sistema satisfaz uma especificação e, se não, o verificador mostra o motivo da especificação ser falsa gerando o contra-exemplo. Muitos sistemas foram verificados com o SMV e fornecem evidências de que o SMV pode ser usado para eliminar erros em projetos reais (BAIER; KATOEN, 2008).

A ferramenta NuSMV consegue reconhecer propriedades tanto em LTL quanto em CTL. Alguns exemplos são: A fórmula EFp que requer que exista algum caminho (E) que eventualmente no futuro satisfaça p, a fórmula AGp que requer que a condição p seja sempre, ou globalmente, verdadeira em todos os estados de todos os caminhos possíveis, e a fórmula EGp que requer que haja algum caminho ao longo da qual condição p é continuamente verdadeira (Cavada R. et al., 2015).

O NuSMV pode lidar com análises qualitativas utilizando as linguagens CTL e LTL. Para lidar com a medição quantitativa, o NuSMV usa uma versão estendida do CTL, o RTCTL (*Real Time CTL*). Apesar de ser capaz de realizar análise quantitativa, o NuSMV não possui o sistema de relógio e não pode definir a restrição de relógio em cada estado, ou seja, ele não possui uma expressividade suficiente para modelar propriedades de tempo real (ZHONGSHENG; XIN; XIAOJIN, 2018).

2.5.2 SPIN

O SPIN é um sistema de verificação genérico que suporta o projeto e a verificação de sistemas de processos assíncronos. Este verificador de modelo aceita as especificações de projeto escritas na linguagem de verificação PROMELA (uma Meta Linguagem de Processo).

Os validadores que são produzidos pela SPIN estão entre os programas mais rápidos para buscas exaustivas e podem ser usados em dois modos diferentes (HOLZMANN, 1991): para modelos de tamanho pequeno a médio com um espaço de estados exaustivo. O resultado de todas as validações realizadas neste modo é equivalente a uma prova exaustiva de correção, para os requisitos de correção que foram especificados.

Para sistemas maiores, os validadores também podem ser usados no modo *"supertrace"*, com a técnica de espaço de estados de bits. Nesses casos, as validações podem ser realizadas em quantidades muito menores de memória.

A linguagem *Promela* é baseada em comunicação ponto a ponto. Assim, quando dois nós querem se comunicar, eles usam um canal predeterminado, *chan broadcastX*, que faz a conexão entre eles(Renesse; Aghvami, 2004).

As fórmulas LTL são usadas para especificar propriedades que serão verificadas. Em SPIN a lógica LTL é expressada da seguinte forma (Lovengreen H., 2016):

a) $[\,]p$: Sempre p;

b) $<> p$: Algum ponto no futuro p;

c) pUq : p é verdadeiro até que q torne-se verdadeiro

2.5.3 UPPAAL

Can et al. (2014) propuseram tanto para a modelagem do sistema quanto para a verificação, a utilização da ferramenta UPPAAL.

UPPAAL não é apenas adequado para a verificação automática de propriedades de segurança e limites, mas também apropriado para sistemas que podem ser modelados como uma coleção de processos não determinísticos com estrutura de controle finito e relógios reais, comunicando através de canais ou variáveis compartilhadas (CAN et al., 2014).

Segundo Can et al. (2014), o simulador em UPPAAL é uma ferramenta de validação que permite o exame de possíveis execuções dinâmicas de um sistema. O verificador de modelo UPPAAL pode verificar as propriedades invariantes e de acessibilidade explorando o espaço de estados de um sistema. Além disso, é permitido declarar variáveis de relógio para registrar o tempo contínuo em UPPAAL.

A ferramenta UPPAAL usa um dialeto da linguagem C como entrada, e é baseada em autômatos temporizados, que são máquinas de estados finitos com tempo (clocks). Um sistema em UPPAAL pode ser composto de processos concorrentes, cada um deles modelados como um autômato.

O autômato tem um conjunto de estados, transições são utilizadas para mudar de estado. Para saber quando fazer uma transição, é possível ter uma guarda e uma sincronização. Um guarda é uma condição e o *clock* diz quando a transição é habilitada. O mecanismo de sincronização é o *hand-shaking*, onde dois processos fazem uma transação no mesmo tempo, duas ações podem ser tomadas, atribuição de variáveis ou reset de clock (UPPAAL, 2009).

As consultas disponíveis no verificador UPPAAL são as seguintes (UPPAAL, 2009):

a) $E <> p$: existe um caminho onde p possivelmente é válido.

b) $A[\,]p$: para todos os caminhos p sempre é válido.

c) $E[\,]p$: existe um caminho onde p é potencialmente válido.

d) $A <> p$: para todos os caminhos, p é eventualmente válido.

e) $p \rightarrow q$: sempre que p for válido, q eventualmente será válido. Essa propriedade também pode ser expressa como: $A\,[\,](p\ imply\ A <> q)$

f) $A\,[\,]\,not\,deadlock$ que verifica se há deadlocks

Alguns exemplos de propriedades são demonstrados abaixo (BEHRMANN; DAVID; LARSEN, 2004):

a) Acessibilidade: $E <> p$ - Dado um estado P, existe um caminho onde P é possivelmente satisfeito.

b) Segurança: $A[]p$ - Dado um estado p, ele precisa ser satisfeito em todos os caminhos.

c) Vivacidade: $A <> p$ - Dado um estado p, o mesmo é eventualmente satisfeito em todos os caminhos.

2.5.4 Time Petri Net (TPN)

As redes de Petri foram utilizadas na descrição formal e análise de sistemas concorrentes, no entanto as redes Petri não lidam de maneira explícita e quantitativa com o tempo, o que as torna inadequadas para a modelagem e especificação de sistemas rigorosos de tempo real. Por esse motivo as redes de Petri foram estendidas para permitir a descrição de fenômenos dependentes do tempo.

As duas extensões de tempo são *Time Petri Nets (TPN)* (Merlin; Farber, 1976) e *Timed Petri Nets* (RAMCHANDANI, 1974). A principal diferença é que enquanto uma transição pode ser disparada dentro de um determinado intervalo na *Time Petri Net*, em *Timed Petri Nets*, as transições são disparadas assim que possível.

O Roméo e TINA são duas das ferramentas que implementam um analisador para a *Time Petri Net (TPN)*. Os Verificadores TINA e Roméo utilizam uma lógica temporal estendida do CTL, o TCTL, para para expressar as propriedades que serão analisadas do modelo (GARDEY et al., 2005)(BERTHOMIEU; RIBET; VERNADAT, 2004).

Os operadores e conectores são descritos na Tabela 2.3 (LIME et al., 2009).:

Tabela 2.3 – Operadores Temporais - TPN

U	Até
E	existe
A	para todos
F	eventualmente
G	sempre
$\rightarrow$	resposta
a, b	parâmetro ou inteiro

Fonte: Elaborado pelo Autor.

De acordo com os operadores descritos Tabela 2.3, é possivel obter propriedades como as que serão definidas a seguir:

a) $EpU[a, b]q$ significa: é possível alcançar, entre a e b unidades de tempo, um estado tal que q é verdadeiro até que p seja verdadeiro.

b) $AG[\,a,b\,]q$: q sempre é válido entre a e b unidades de tempo.

c) $EG[\,a,b\,]q$: Existe um caminho no qual q sempre é verdadeiro (entre a e b unidades de tempo).

d) $p \rightarrow [\,0,b\,]q$: quando p é verdadeiro, então, q será verdadeiro dentro de b unidades de tempo.

2.5.5 TLA$^+$

TLA$^+$ é uma linguagem baseada na lógica Temporal de Ações (TLA) e é uma linguagem de especificação para descrever sistemas concorrentes assíncronos, não determinísticos. A TLA foi desenvolvida para permitir a formalização mais simples e direta de provas de assertividade de sistemas concorrentes. Uma especificação escrita nesta linguagem é uma fórmula que descreve uma máquina de estado em termos de uma condição inicial, uma relação de estado seguinte e possivelmente algumas condições de atividade (YU; MANOLIOS; LAMPORT, 1999).

Yu, Manolios e Lamport (1999) criaram o verificador de modelos TLC que é capaz de lidar com a classe TLA$^+$. O verificador de modelo TLC inclui a maioria das especificações de projetos de sistemas reais, no entanto, pode não ser capaz de lidar com um modelo suficientemente grande, de forma eficiente para detectar erros, exceto os mais simples. Além disso, o TLC não pode verificar propriedades de vivacidade (LAMPORT, 2002).

Existe ainda o sistema de provas TLA - TLAPS, em desenvolvimento pelo *Microsoft Research- INRIA*. O TLAPS pode trabalhar em conjunto com verificador TLC, de forma que o verificador de modelos TLC encontra rapidamente pequenos erros antes do início da verificação, e o TLAPS por sua vez, verifica propriedades do sistema que estão além dos recursos da verificação finita de modelos. No entanto, a versão atual do TLAPS não executa o raciocínio temporal e não trata de alguns recursos do TLA$^+$ (MICROSOFT, 2020).

Para especificar propriedades temporais, em TLC é utilizado a seguinte sintaxe (WAYNE, 2018):

a) $[\,]P$: significa que P é verdadeiro para todos os estados. Em outras palavras, um invariante. o TLC interpreta como uma propriedade temporal $[]P$.

b) $<>$: significa eventualmente : $<> P$ significa que para cada comportamento possível, pelo menos um estado tem P como verdadeiro.

c) $\rightarrow$ significa "implica": $P \rightarrow Q$ implica que se P alguma vez se tornar verdadeiro, em algum ponto depois, Q deve ser verdadeiro.

2.5.6 Comparativo entre as Linguagens

A Tabela 2.4, apresenta um breve comparativo entre as ferramentas de verificação de modelos apresentadas nesse capitulo:

Tabela 2.4 – Quadro comparativo de linguagens.

Ferramenta	Geração de contra exemplo	Licença de software	Ling. de entrada	ST/ Modelos	Ling. Prop
NuSMV	Sim	Livre	SMV	Simples	CTL, LTL, PSL
UPPAAL	Sim	Livre para acad.	C	Tempo Real	TCTL
SPIN	Sim	Livre para acad.	Promela	Simples	LTL
ROMEO,TINA	Sim	Livre	Time Petri Nets	Tempo Real	TCTL
TLC	Sim	Livre	TLA+	Tempo Real	TLA

Fonte: Elaborado pelo Autor

A verificação de modelos ainda apresenta algumas limitações, uma delas é que é possível verificar apenas os requisitos estabelecidos, ou seja, não há garantia de completude. A validade de propriedades que não estão verificadas não pode ser julgada.

No entanto, os estudos de caso apresentados nos trabalhos considerados, demonstram que modelos complexos podem ser verificados com o auxílio dos verificadores de modelo. Através da verificação de modelos é possível reduzir consideravelmente os erros que geralmente aparecem no início do projeto. Através dos estudos levantados, foi possível perceber que já existem ferramentas de verificação de modelos implementadas que se mostram eficientes para a realização de tal tarefa.

Para esse trabalho, serão considerados os verificadores mais apropriados par sistemas concorrentes. As linguagens apresentadas pelo SPIN e NuSMV não serão consideradas pois não são capazes de abordar propriedades quantitativas.

A Tabela 2.5, apresenta um comparativo entres as linguagens e ferramentas capazes de verificar sistemas concorrentes, levando em consideração as principais características desejadas para a realização desse trabalho.

Tabela 2.5 – Quadro comparativo - Características de linguagens.

Ferramenta	TLA+	UPPAAL	TINA
Verificação de Vivacidade (Liveness)	Limitado	Suporta	Suporta
Verificação de Deadlock	Suporta	Suporta	Suporta
Verificação de Segurança (Safety)	Suporta	Suporta	Suporta
Modelagem de tempo da transição	Suporta	Suporta	Suporta
Modelagem de variaveis de relógio (clock)	Suporta	Suporta	Não Suporta
Sincronismo por canal	Suporta	Suporta	Não Suporta

Fonte: Elaborado pelo Autor.

As seguintes características podem ser verificadas pela ferramenta de análise TINA: existência de estado terminal, segurança (*safety*), vivacidade (*Liveness*), reversibilidade (*reversibility* - verificar se a rede de Petri voltará ao seu estado inicial em qualquer estado que atingir), consistência (*consistency*) (GHOMARI; DJERABA, 2010). Em relação a modelagem de sincronismo com canais, existem estudos relacionados ao tema como o estudo apresentado por Serugendo et al. (2002). No entanto, a ferramenta de modelagem TINA não é capaz de expressar esse conceito.

O verificador de modelos TLC para TLA^+ apresenta limitações em sua capacidade de lidar com propriedades de vivacidade (*Liveness*). Segundo a documentação encontrada em Microsoft (2020), pode ser impossível descobrir uma violação de uma propriedade de vivacidade (*Liveness*) em um modelo finito.

Já o UPPAAL apresenta características importantes para verificação de sistemas de concorrentes, que é a modelagem de sincronização através de canais. O UPPAAL possibilita a construção do modelo físico do sistema sem a necessidade do uso de ferramentas auxiliares. UPPAAL ainda utiliza como linguagem temporal a TCTL que segundo Farn Wang (2004) a *Timed Computational Tree Logic (TCTL)* é a lógica temporal mais usada para sistemas em tempo real. Todas as três ferramentas são capazes de detectar ausência de *deadlock* no processo de verificação.

Para que seja possível aplicar a verificação de modelos, o sistema deve ser modelado em linguagem formal, além disso, é necessário que os requisitos do sistema sejam traduzidos para uma lógica temporal. Os requisitos do sistema são geralmente expressos em linguagem natural, no caso deste trabalho, o Português. Considerando o comparativo apresentado na Tabela 2.5, esse trabalho considera a linguagem UPPAAL como linguagem alvo da tradução.

2.6 ESPECIFICAÇÕES DE REQUISITOS

A elicitação e especificação de requisitos é um processo fundamental no ciclo de vida de um software. Os requisitos definem o comportamento do software. O SWEBOK (BOURQUE; RICHARD, 2014), afirma que os requisitos são responsáveis por expressar as necessidades e as restrições de um produto.

Os requisitos devem ser declarados de forma clara e não ambígua. Requisitos vagos e não verificáveis que dependem de um julgamento sobre sua interpretação como “O software deve ser confiável”, devem ser evitados (BOURQUE; RICHARD, 2014).

Segundo ISO/IEC/IEEE 29148-2018(IEEE, 2018), ao se escrever requisitos, é necessário se atentar a construção textual desses requisitos. Os requisitos devem dizer o que o sistema de software deve fazer e não como ele deve fazer.

Ao se escrever requisitos, os termos vagos e gerais devem ser evitados. A utilização desses termos podem gerar requisitos não verificáveis. Além disso, a utilização de termos vagos, pode gerar requisitos ambíguos. Um requisito ambíguo permite múltiplas interpretações por parte dos envolvidos no projeto (IEEE, 2018).

O padrão ISO/IEC/IEEE 29148-2018 (IEEE, 2018) fornece uma lista de termos considerados vagos:

- superlativos : melhor, mais;
- linguagem subjetiva: amigável, fácil de usar, econômico;
- pronomes vagos: ele, isto, aquilo;
- termos que são ambíguos como advérbios e adjetivos: quase sempre, significativa, mínimo;
- termos não verificáveis: fornece suporte, mas não limitado para;
- frases comparativas: melhor que, maior qualidade;
- brechas como: se possível, conforme apropriado, como aplicável;
- termos que sugerem totalidade: todos, sempre, nunca, tudo.

O processo de tradução de requisitos escritos em linguagem natural - Português, com o objetivo gerar uma propriedade temporal requer a aplicação do processamento de linguagem natural (PLN), a qual será abordada na próxima seção.

2.7 PROCESSAMENTO DE LINGUAGEM NATURAL (PLN)

O processamento de Linguagem natural (PLN) surgiu em 1950, como resultado de estudos na área de inteligência artificial e linguística. Esse processamento

tem como objetivo estudar o desenvolvimento de programas de computador que analisam, reconhecem e geram textos em linguagens humanas, ou linguagens naturais (KHURANA et al., 2017).

O PLN não é uma tarefa simples, isso pois a linguagem natural é ambígua. Essa ambiguidade torna o PLN mais complexo que o processamento das linguagens de programação, visto que as linguagens de programação são formalmente definidas. Isso evita a ambiguidade. Segundo Taskin e Al (2019), alguns objetivos usuais em PLN são recuperação de informação a partir de textos, tradução automática, interpretação de textos e realização de inferências a partir de textos.

As técnicas de PLN podem ser aplicadas em diversas áreas, e algumas delas são descritas a seguir (OTHERO, 2006):

- Classificação de Textos;
- Sistemas de perguntas e respostas;
- Extração de informações;
- Tradução automática.

O trabalho em questão aborda o processo de tradução entre duas linguagens, A linguagem natural - Português - e linguagem formal - propriedades de lógica temporal. O processo de tradução automática será abordado nas próximas seções.

2.8 TRADUÇÃO AUTOMÁTICA (TA)

Tradução automática, do inglês *Machine Translation*, é um campo do processamento de linguagem natural (PLN) que investiga como usar software para traduzir texto ou fala de um idioma para outro sem envolvimento humano. A premissa da TA e de que a partir de uma sentença fornecida como entrada na linguagem de origem, um sistema gera uma sentença equivalente em uma linguagem alvo. A pesquisa no campo de TA se iniciou em 1950, e apresenta três linhas principais de desenvolvimento, a tradução automática baseada em regras, a tradução automática estatística e a tradução automática neural (YANG; WANG; CHU, 2020a). Nesse trabalho serão abordados os seguintes métodos de tradução PLN (YIN et al., 2017):

a) Tradução Automática Baseada em Regras: Análise segmentada, conhecida como *parsing*. Nessa abordagem as regras são elaboradas manualmente.

b) Tradução Automática Neural.

Esses métodos serão apresentados nas próximas seções.

2.8.1 Tradução automática neural (TAN)

A tradução automática neural apresenta como vantagem a pouca necessidade de conhecer regras de linguística para construção do tradutor, no entanto a TAN apresenta uma alta complexidade no treino das redes neurais. (Luong et al. (2016))

A tradução automática neural é realizada utilizando redes neurais. O *Deep Learning* ou aprendizagem profunda, ganhou força com os ótimos resultados apresentados em tarefas de PLN (KALCHBRENNER; BLUNSOM, 2013).

Um estudo realizado por Singh et al. (2021) comparou a eficiência entre os modelos: *Naive Bayes classifier, Extreme Gradient Boosting (XGBoost) classifier, SVM, Long Short Term Memory Networks (LSTM), e Convolution Neural Network (CNN)*. Os modelos foram aplicados no contexto de processamento de linguagem natural utilizando uma mesma base de dados.

Ao analisar os resultados foi observado que as redes LSTM e CNN se destacaram entre as demais. Ao usar um LSTM, todo o contexto de uma sentença pode ser processado pela rede neural, incluindo dependências longas e intervalos de tempo do passado ou futuro. Já o resultado das CNNs se deve ao fato de que os vetores de entrada são representados com mais detalhes, quando passados pelas camadas do modelo (SINGH et al., 2021).

Para compreender a tradução baseada em rede neural os conceitos sobre redes neurais artificiais (RNAs) serão descritas nas próximas seções.

2.8.1.1 Redes neurais artificiais (RNAs)

As redes neurais artificiais (RNAs) são métodos para aprendizado que trabalham a partir de exemplos fornecidos. As RNAs são compostas por *neurônios* que são modelos matemáticos compostos por uma combinação linear entre o vetor de entrada, o vetor de *pesos* e o *bias* (limiar de ativação). O resultado dessa combinação linear é passado como argumento para uma *função de ativação* resultando na saída final do neurônio.

Uma RNA é formada por *camadas*, no qual a primeira camada é denominada camada de entrada, as camadas intermediárias são denominadas camadas ocultas e a última camada é a camada de saída. Cada camada recebe os valores da camada anterior como entrada (HAYKIN, 2007).

O processo de aprendizado de uma rede é definido sobre um conjunto de regras e nesse processo existem alguns conceitos que serão apresentados a seguir (BRAGA, 2007):

1. Tipo de Aprendizado:
    a) Supervisionado: Quando as respostas desejadas são fornecidas a rede, e a rede deve aprender a relação entre o padrão de entrada e o de

resposta. Nesse tipo de aprendizado as respostas são previamente fornecidas a rede.

b) Não Supervisionado: Quando as respostas desejadas não são fornecidas para todas as entradas. São fornecidos apenas exemplos para que a rede realize um treino.

2. Correção de pesos:

   a) Modo Padrão: A correção acontece a cada apresentação de um exemplo do conjunto de treino. Para cada ciclo, em cada exemplo, é realizada a correção do peso.

   b) Modo Batch: Todos os exemplos de treino são apresentados a rede, que calcula um erro médio. Uma correção ocorre por ciclo.

3. Épocas: Quantidade de ciclo de treinamento completo de uma rede.

4. Função de ativação: A função de ativação é utilizada pra calcular a saída do neurônio. A função de ativação é necessária pois a soma ponderada pode resultar em um valor qualquer, pois a previsão é uma probabilidade. A função de ativação converte o valor, em um valor entre 0 e 1.

   Entre as Funções de ativação estão as seguintes funções: função logística, tangente hiperbólica e a unidade linear retificada (*ReLU*).

A rede *feedforward* foi a primeira e também mais simples tipo de rede neural artificial desenvolvida, uma breve introdução dessa rede será abordada na próxima seção.

2.8.1.2 Rede neural direta (*feedforward*)

As redes *feedforward* alimentam informações diretamente, ou seja, as informações trafegam em apenas uma direção (HAYKIN, 2007). Uma das redes mais conhecidas dessa classe é o perceptron de camada única que consiste em uma única camada de nós de saída.

As entradas são alimentadas diretamente nas saídas por meio de uma série de pesos. A soma dos produtos dos pesos e das entradas é calculada em cada nó, e se o valor estiver acima de algum limiar, normalmente 0, o neurônio dispara e assume o valor ativado, caso contrário, ele assume o valor desativado(HAYKIN, 2007).

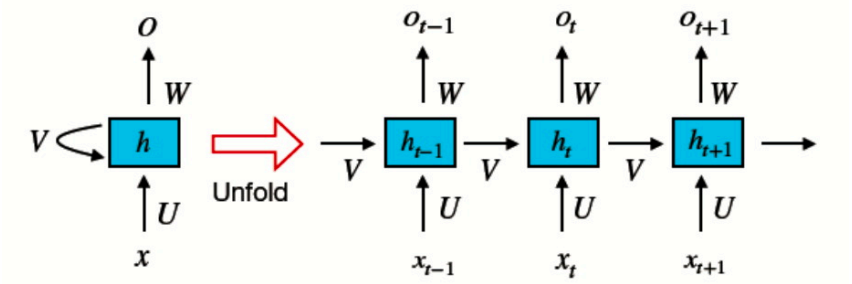
Uma rede *feedforward* não tem noção de ordem no tempo, e a única entrada que considera é o exemplo atual a que foi exposta. As redes feedforward são amnésicas em relação ao seu passado recente (HAYKIN, 2007).

Diferente da rede *feedforward*, as redes neurais recorrentes possuem realimentação e utilizam o estado anterior em conjunto com o estado atual para gerar as saídas. As redes neurais recorrentes serão abordadas na próxima seção.

2.8.1.3 Redes neurais recorrentes (RNN)

A rede neural recorrente (RNN) abordada na Figura 2.9 visam realizar o processamento de forma que, um passo receba como entrada o resultado do processamento do passo anterior. Essa abordagem tem duas fontes de entrada que se combinam para determinar o comportamento dos dados (ELMAN, 1990).

Figura 2.9 – Estrutura da rede recorrente



Fonte: LeCun, Bengio e Hinton (2015, p. 2).

O diagrama da Figura 2.9, mostra um desdobramento da rede RNN. Um pedaço da rede neural olha para alguma entrada x e produz um valor O. Um *loop* permite que as informações sejam passadas de uma etapa da rede para a próxima etapa. As fórmulas que estruturam o cálculo de uma RNN são as seguintes:

- x_t é a entrada no passo de tempo t.
- h_t é o estado oculto no passo de tempo t. É a "memória" da rede. O termo h_t é calculado com base no estado oculto anterior e a entrada na etapa atual através da fórmula: $ht = f(Ux_t + Wh_{t-1})$. A função geralmente é uma não-linearidade, como tanh ou ReLU. Já $h - 1$, que é necessário para calcular o primeiro estado oculto, é tipicamente inicializado com zero.
- O_t é a saída na etapa t. Por exemplo, para prever a próxima palavra em uma frase, seria um vetor de probabilidades do vocabulário: $O_t = softmax(Vh_t)$.

O estado oculto h_t é como a memória da rede. h_t captura informações sobre o que aconteceu em todas as etapas de tempo anteriores. A saída na etapa O_t é calculada exclusivamente com base na memória no tempo t. Uma RNN compartilha os mesmos parâmetros (U, V, W) em todas as etapas o que diz que a mesma tarefa é executada em cada etapa, apenas com entradas diferentes.

Segundo Young et al. (2017) as redes neurais recorrentes são utilizadas para modelar problemas de PLN devido sua boa performance e capacidade de capturar as características sequenciais presentes na linguagem natural. Assim como os humanos compreendem as palavras, frases e sentenças baseados na compreensão adquirida

ao longo do tempo, as redes recorrentes funcionam com *loops* permitindo que as informações anteriores sejam consideradas durante o processamento.

Modelos baseados em redes neurais recorrentes (RNN) assumiram a posição dominante no desenvolvimento de tradutores automáticos baseados em rede neural alcançando um desempenho satisfatório (YANG; WANG; CHU, 2020b). O primeiro tradutor baseado em RNN bem sucedido foi proposto por Sutskever, Que utilizou um modelo RNN puro. (SUTSKEVER; VINYALS; LE, 2014b). Os modelos baseados em Rede Neural Convolucional (CNN) não tiveram desempenho competitivo com o modelo baseado em RNN (YANG; WANG; CHU, 2020b).

A ideia sobre as redes RNNs é de que elas são capazes de conectar informações passadas à uma tarefa atual. Para algumas tarefas é necessário apenas olhar as informações passadas recentes para realizar a tarefa atual. Nos casos onde a lacuna entre as informações relevantes é pequena, os RNNs podem aprender a usar as informações anteriores (GRAVES, 2012).

No entanto, há casos em que são necessárias informações sobre o contexto. Nesses casos a lacuna entre as informações relevantes e o ponto onde elas são necessárias pode ser muito grande. A medida em que essa lacuna aumenta, os RNNs tornam-se incapazes de aprender a conectar as informações. Para tentar sanar esse problema, surgiram as redes de memória longa de curto prazo conhecidas como LSTM (Long Short Term Memory) (GRAVES, 2012).

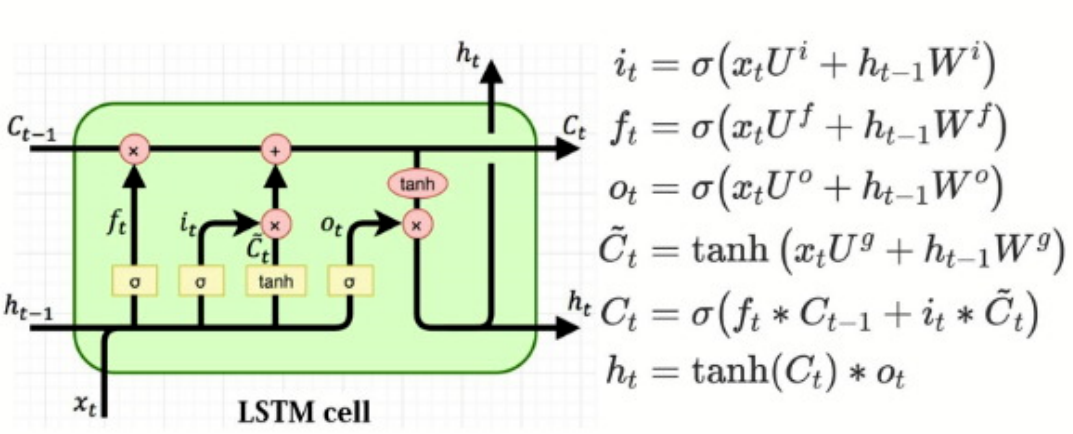
2.8.1.4 Redes de Memória Longa de Curto Prazo (LSTM)

A LSTM ou Rede de memória longa de curto prazo, é uma arquitetura de rede neural recorrente (RNN) capaz de processar e prever séries temporais com longos intervalos. Em uma RNN tradicional, se for necessário prever uma sequência após 500 intervalos provavelmente o modelo não seria capaz de utilizar a informação contida no ponto de partida, no entanto o LSTM é capaz de utilizar as informações contidas nesse ponto, pois possuem o conceito de células de memória (HOCHREITER; SCHMIDHUBER, 1997).

Os LSTMs são projetados explicitamente para evitar o problema de dependência de longo prazo. O módulo de repetição em um LSTM contém quatro camadas de interação que pode ser vista na Figura 2.10.

Em uma célula LSTM, há portões extras, ou seja, portões de entrada, esquecimento e saída que são usadas para decidir quais sinais serão encaminhados para outro nó. As equações que descrevem o comportamento de todos os portões na célula LSTM são descritas na Figura 2.10. O $\mathbf{x_t}$ é o *input* da rede no momento t, $\mathbf{h_t}$ é o *output* da célula no instante t, σ é a função de ativação logística, $\mathbf{C_t}$ denota o estado da célula no momento t, $\tilde{\mathbf{C}}_\mathbf{t}$ representa o candidato de estado para o instante t (VARSAMOPOULOS; BERTELS; ALMUDEVER, 2020).

Figura 2.10 – Estrutura da rede LSTM



Fonte: Varsamopoulos, Bertels e Almudever (2020, p. 4).

Adicionalmente existem 3 portões nas células LSTM que são: portões de esquecimento $\mathbf{f_t}$, portões de entrada $\mathbf{i_t}$ e o portão de saída $\mathbf{o_t}$. As constantes $\mathbf{W^i}$, $\mathbf{W^f}$,$\mathbf{W^o}$,$\mathbf{W^g}$ são os pesos dos portões de esquecimento, entrada, saída e da própria célula.

A informação de entrada será memorizada na célula ou não, essa memorização é uma decisão tomada pelo portão de entrada. O portão de saída define se a informação será descartada no momento t (VARSAMOPOULOS; BERTELS; ALMUDEVER, 2020). Resumidamente, esse processo define se as entradas são relevantes ou se irão prejudicar e portanto devem ser ignoradas na atualização do conteúdo da memória interna.

Sutskever, Vinyals e Le (2014a) propuseram um abordagem baseada em uma arquitetura codificador-decodificador utilizando as redes LSTM. Essa arquitetura foi utilizada pelo google em aplicações de tradução automática (WU et al., 2016).

2.8.1.5 Codificador-Decodificador

A estrutura Codificador-Decodificador contém duas redes conectadas em sua arquitetura, quando a rede do codificador recebe uma frase de origem, ela lê a frase de origem, palavra por palavra e compacta a sequência de comprimento variável em um vetor de comprimento fixo em cada estado oculto. Este processo é chamado de codificação (BAHDANAU; CHO; BENGIO, 2014).

Dado o estado oculto final do codificador, conhecido como vetor de pensamento, o decodificador irá fazer o trabalho inverso, transformando o vetor de pensamento na frase-alvo, palavra por palavra. A estrutura do codificador-decodificador direciona a tarefa de tradução dos dados de origem diretamente para o resultado de destino, o que significa que não há resultado visível no processo médio, isso também é chamado de tradução de ponta a ponta (BAHDANAU; CHO; BENGIO, 2014).

As redes codificador-decodificador apresentaram limitações ao lidar com alguns

casos de tradução visto que essas redes usam um tamanho fixo para a representação da entrada e saída. No entanto, o tamanho da entrada e saída podem ser diferentes para algum contexto.

Além disso essa abordagem pode apresentar problemas ao traduzir sequências muito longas, pois o vetor de contexto precisa considerar a informação de toda a sequência de entrada, ou seja, ele teria que ser capaz de representar uma longa dependência no tempo, das palavras de entrada e saída. O mecanismo de atenção surgiu como uma alternativa para suprir essa limitação (BAHDANAU; CHO; BENGIO, 2014).

O mecanismo de atenção fica localizado entre o codificador e o decodificador, sua entrada é composta pelos vetores de saída do codificador e os estados do decodificador. A saída de atenção é uma sequência de vetores chamados vetores de contexto. Os vetores de contexto permitem que o decodificador se concentre em certas partes da entrada ao prever sua saída, ou seja, o decodificador utiliza todos os estados intermediários para gerar a saída (BAHDANAU; CHO; BENGIO, 2014).

A arquitetura *transformer* baseada principalmente no mecanismo de atenção foi desenvolvida pelo Vaswani et al. (2017) e será apresentada na próxima seção.

2.8.1.6 Transformer

Transformer é uma arquitetura de modelo que evita a recorrência e, em vez disso, depende inteiramente de um mecanismo de atenção para desenhar dependências globais entre a entrada e a saída (VASWANI et al., 2017).

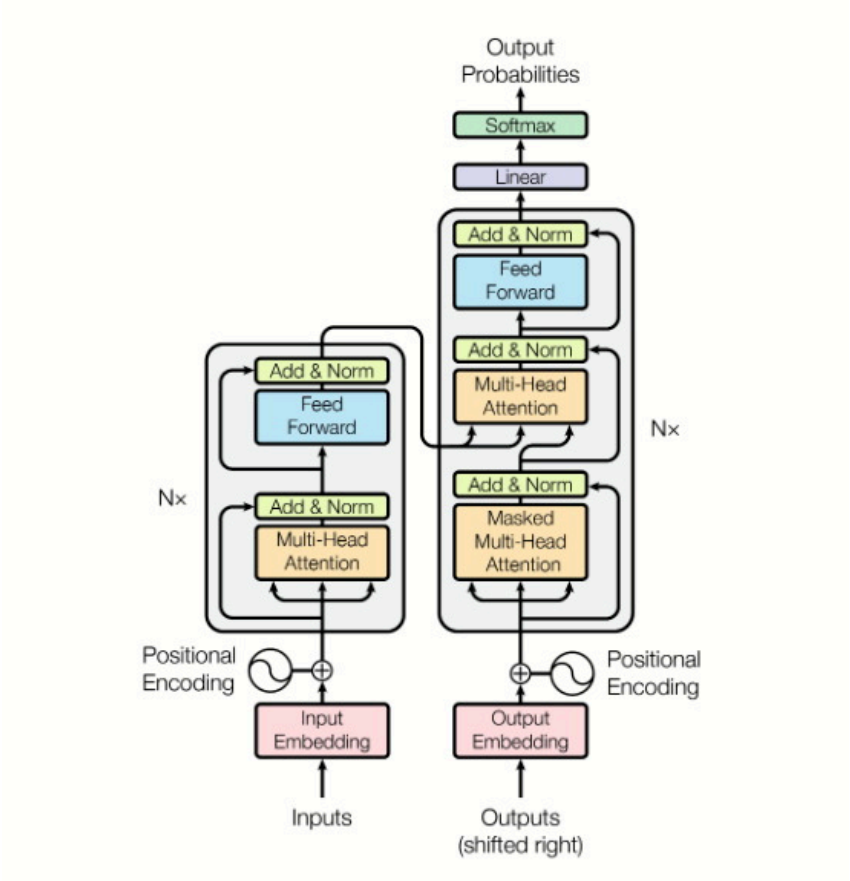
O codificador mapeia uma sequência de entrada de representações de símbolo $(x_1, ..., x_n)$ para uma sequência de representações contínuas z = $(z_1, ..., z_n)$. Dado z, o decodificador então gera uma sequência de saída $(y_1, ..., y_m)$ de símbolos, um elemento por vez. Em cada etapa, o modelo é auto-regressivo, consumindo os símbolos gerados anteriormente como entrada adicional ao gerar o próximo(VASWANI et al., 2017). A Figura 2.11 apresenta a arquitetura da Rede *transformer*:

O codificador, apresentado na parte esquerda da Figura 2.11 é composto por uma pilha de N = 6 camadas idênticas. Cada camada possui duas subcamadas. O primeiro é um mecanismo de auto-atenção e o segundo é uma rede de alimentação direta (*feedforward*) totalmente conectada.

O decodificador apresentado na parte direita da Figura 2.11 também é composto por uma pilha de N = 6 camadas idênticas. Além das duas subcamadas em cada camada do codificador, o decodificador insere uma terceira subcamada, que executa a atenção múltipla sobre a saída da pilha do codificador.

Para a construção do codificador e do decodificador são utilizados os seguintes conceitos :

Figura 2.11 – Transformer



Fonte: Vaswani et al. (2017, p. 3).

- *Scaled Dot-Product Attention*: Também chamada de atenção particular a *Scaled Dot-Product Attention* é calculada de acordo com a Equação 2.1:

$$Attention(Q, K, V) = softmax(QK^T/\sqrt{d_k})V \tag{2.1}$$

onde d_k é a dimensão das entradas, Q é o vetor de busca, K o vetor de chaves, e V o vetor de valores.

- *Multi-head attention*: Permite que o modelo atenda conjuntamente às informações de diferentes subespaços de representação em diferentes posições e poder ser calculado conforme as Equações 2.2 e 2.3 :

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2.2}$$

onde

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.3}$$

O *Multi-head attention* é utilizado nas camadas de atenção do codificador-decodificador, onde as consultas vêm da camada do decodificador anterior e as chaves de memória e os valores vêm da saída do codificador. Isso permite

que cada posição no decodificador atenda a todas as posições na sequência de entrada. Também é utilizado no codificador e no decodificador separadamente.

- O codificador e decodificador contém em uma de suas camadas, a rede *feed-forward* totalmente conectada. A rede consiste em duas transformações lineares com uma ativação ReLU entre elas. A Equação 2.4 representa o cálculo para a rede *feedforward*:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.4}$$

O *transformer* é um modelo de transição de sequência baseado inteiramente na atenção. Assim esse mecanismo de atenção substitui as camadas recorrentes mais comumente usadas em arquiteturas de codificador-decodificador. O *transformer* pode ser treinado significativamente mais rápido do que as arquiteturas baseada em camadas recorrentes ou convolucionais (VASWANI et al., 2017).

2.8.2 Tradução automática baseada em regras

A abordagem da tradução baseada em regras visa a vinculação da estrutura da sentença de entrada com a estrutura da sentença de saída exigida, preservando seu significado (NIRENBURG; WILKS, 2000).

As regras da tradução podem ser aplicadas nos níveis léxico, sintático ou semântico. Caso as regras sejam aplicadas apenas no nível léxico, essa categoria de tradução é classificada como uma tradução direta (MORRISSEY, 2008) e o resultado é uma substituição das palavras da sentença de origem para as correspondentes na estrutura da sentença de destino, fazendo com que a estrutura gramatical seja mantida.

Por exemplo, uma sentença em português que será traduzida para o inglês:“João comprou duas maçãs”.

Na tradução baseada em regras algumas informações devem ser fornecidas, como:

- Um mapeamento das palavras em português e sua correspondente em inglês.
- Regras que representem estruturas das frases em inglês.
- Regras que representam estruturas das frases em português.

A partir desse conjunto de informações é possível criar regras capazes de relacionar uma sentença na linguagem de entrada para a sentença na linguagem de saída. Para obter informações básicas da classe gramatical de cada palavra fonte é necessário realizar uma análise léxica, através dessa análise é possível obter a estrutura da sentença: João = substantivo próprio, comprou = verbo, duas = numeral cardinal, maças = substantivo.

2.8.2.1 Análise léxica e sintática

A análise léxica permite verificar um alfabeto criado pelo usuário ou um alfabeto já existente. O analisador léxico, verifica a sequência de entrada e traduz essa sequência em símbolos léxicos ou *tokens*.

Os *tokens* são uma classe de símbolos e podem ser representados por um conjunto, como uma instância de uma mesma unidade léxica. Na linguagem natural cada *token* pode ser definido como uma palavra ou símbolo (GESSER; FURTADO, 2002).

Se um analisador léxico de linguagem de programação estivesse tentando reconhecer os seguintes dados: "*while(i < 10) i++*". A interpretação do analisador provavelmente seria de que a *string "while"* representa a identificação de uma condição, *"i"* representa o nome de uma variável, "10" representa um numeral, "<" e "++" como operadores (ROCHE, 1997).

Os analisadores léxicos de textos em linguagem natural apresentam semelhanças com os analisadores léxicos de linguagem de programação. Por exemplo, se o analisador recebe a seguinte entrada: "*O homem comeu as maçãs.*" O objetivo do analisador é identificar as unidades léxicas que constituem a sentença e classificá-las em grupos gramaticais (ROCHE, 1997). Na análise da sentença acima pode-se identificar através da definição léxica abaixo:

a) **Det** $\rightarrow$ **O, A, Os, As**

b) **Subs** $\rightarrow$ **homem, mulher, maçãs.**

c) **Verb** $\rightarrow$ **comer, beber**

Tem-se que: "O" é um determinante artigo definido, "homem" é um substantivo, "comeu" como o verbo, "as" como determinante artigo definido e "maçãs" como um substantivo (ROCHE, 1997).

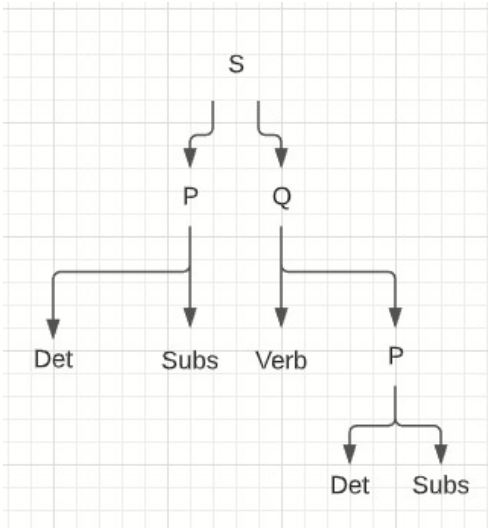
O analisador léxico inicia a verificação e define o estado inicial como o estado atual, então ele obtém o próximo símbolo da sentença. Caso não exista um próximo símbolo, o analisador define o estado como sendo o estado final. Se todos os símbolos da sentença são reconhecidos, o analisador retornará um valor verdadeiro.

Na análise, os caracteres são agrupados em *tokens*, no entanto, nem todos os caracteres da entrada geram *tokens*. Espaços em branco e quebra de linha são ignorados por não conter nenhuma informação relevante (GESSER; FURTADO, 2002).

Já na análise sintática, é feito uma verificação sobre a sentença de origem a partir dos resultados obtidos na análise léxica. Dessa forma é possível verificar se a sentença de origem está em concordância com a gramática da linguagem de origem. Como no exemplo: "*O homem comeu as maçãs.*" onde a concordância gramatical possui a estrutura mostrada na Figura 2.12.

Após essa análise, pode ser aplicado um processamento de adaptação onde as

Figura 2.12 – Estrutura Gramatical



Fonte: Elaborado pelo Autor

regras de construção gramatical da linguagem de destino são aplicadas na sentença de entrada. Nessa etapa pode se aplicar uma modificação de palavras, ou uma remoção de classes gramaticais como artigos definidos (as, os, a, o), alterações de verbos e até alterações na ordem de uma sentença.

2.8.3 Considerações

A tradução automática apresenta alguns desafios visto que a linguagem natural é ambígua e uma única frase pode ser interpretada de várias maneiras distintas. Além disso, o significado de algumas sentenças pode depender do contexto sendo necessário analisar outras sentenças que complementam esse contexto.

A tradução automática baseada em regras e a tradução automática baseada em redes neurais são estratégias que podem ser utilizadas com o intuito de lidar com esses desafios.

A tradução automática baseada em regras apresenta uma vantagem, pois as regras linguísticas são legíveis. No entanto, esse categoria de tradução pode apresentar um alto custo na construção dessas regras. A cobertura desse categoria de tradução é limitada, pois é impossível traduzir uma sentença que não esteja escrita de forma a atender esse conjunto de regras.

Já a tradução automática baseada em redes neurais, diferente da tradução automática baseada em regras, consegue lidar com um baixo conhecimento linguístico, mas pode apresentar uma alta complexidade computacional para o treinamento das redes neurais. Na próxima seção são apresentados alguns trabalhos que utilizam essas duas abordagens de tradução.

2.9 TRABALHOS RELACIONADOS

Apresenta-se nessa seção um conjunto de trabalhos com uma relação com o presente estudo. Esses trabalhos visam a tradução de requisitos escritos em linguagem natural para propriedades temporais formais (não limitados para CTL).

No trabalho de Nelken e Francez (1996), o método de tradução consiste em duas transições principais: de linguagem natural (LN) em inglês para Teoria da Representação do Discurso (DRT) e de DRT para Lógica Temporal (TL). Foi definido um critério de correção apenas para a tradução entre DRT e TL.

O programa SpecTran 1.0 recebe como entrada as propriedades em linguagem natural, analisa, constrói as estruturas em DRT e gera uma fórmula ACTL. O analisador foi escrito usando o sistema LexGram. Em SpecTran, a gramática alemã foi substituída pela inglesa.

A SpecTran analisa as estruturas de entrada frase após frase, processando cada nova frase e atualizando a estrutura. Em casos de ambiguidade, o parser produz todos os parses alternativos. SpecTran consulta o usuário com relação à resolução do significado dos pronomes (por exemplo: isso, eles).

Segundo os autores, ainda é necessário que os *designers* escrevam especificações em linguagem precisa e concreta, mas alguma tolerância é permitida no uso de estrutura sintática flexível, anáfora pronominal e relações temporais inter-sentenciais e inter-sentenciais complexas.

Ao introduzir e provar um critério de correção para a segunda fase da tradução e com base na pesquisa linguística para a primeira parte da tradução, foi possível garantir que a transformação de LN em TL não introduz erros. Essa garantia falta para a tradução manual muitas vezes exercido na prática e de tentativas anteriores de tradução automática (NELKEN; FRANCEZ, 1996).

Em Dwyer, Avrunin e Corbett (1999), os autores apresentam uma abordagem baseada em padrões para a codificação de especificações de propriedade para verificação de sistema de estado finito. Entre outras contribuições, eles fornecem um conjunto de 55 fórmulas baseadas em LTL que devem capturar padrões típicos que entram em jogo no desenho de sistemas concorrentes e reativos, como a existência de uma condição específica ou a resposta condicional a um estímulo.

No trabalho de Dzifcak et al. (2009), um estudo focado na tradução de linguagem natural é apresentado para propriedades temporais em CTL. A tradução é realizada utilizando um analisador. A implementação dessa abordagem foi aplicada em um robô que recebe instruções de linguagem natural em um ambiente de escritório.

Em Raman et al. (2013), o domínio do robô móvel é novamente considerado. Os verbos e seus argumentos são extraídos da árvore sintática, transformando-a em um conjunto de quadros. Em seguida, um conjunto predefinido de padrões LTL é reconhecido utilizando tradução baseada em regras. Os LTLs são associados a tipos

específicos de quadros. as contribuições são ilustradas com exemplos envolvendo um assistente de robô em um hospital.

Instruções informais escritas em linguagem natural são transformadas em especificações formais e usadas para sintetizar um controlador.Para especificações não implementáveis, a análise refinada fornece ao usuário uma explicação concisa das partes da especificação que causam a falha.

No trabalho de Yan, Cheng e Chai (2015), a gramática NL estruturada proposta para LTL não se limita a um domínio particular como na abordagem de Dzifcak et al. (2009). As proposições atômicas são encontradas, e logo após, regras elaboradas à mão são aplicadas à árvore de sintaxe para extrair operadores lógicos e temporais.

No trabalho de Harris e Harris (2015) foi proposto a criação de propriedades de verificação formal a partir de descrições textuais escritas em linguagem natural. São apresentadas duas abordagens que utilizam processamento linguagem natural (PLN) e técnicas de aprendizado de máquina para a geração automática de propriedades.

Na primeira abordagem, um conjunto de propriedades expressas em linguagem natural são divididos em subconjuntos de sentenças estruturalmente semelhantes. Um modelo de propriedade formal generalizado para cada subconjunto é usado para fornecer um mapeamento de cada frase para uma propriedade de verificação bem especificada.

Em uma segunda abordagem é proposta uma gramática formal personalizada que captura a semântica da CTL. Então é implementado um sistema de tradução que usa essa gramática para realizar uma análise semântica nas propriedades escritas em linguagem natural.

Para facilitar a criação da gramática, é proposto um algoritmo de aprendizagem que gera automaticamente gramáticas a partir de um pequeno conjunto de propriedades em linguagem natural e propriedades formais. Esta gramática gerada por máquina é usada no sistema de tradução.

Em Ghosh et al. (2016), o framework ARSENAL faz a tradução de requisitos de linguagem natural em representações formais analisáveis. Dada uma frase de entrada, por exemplo: "Temperatura inferior desejada", essa frase é então convertida em termos únicos como, por exemplo, Temperatura_Inferior_Desejada, na tentativa de regularizar o texto de entrada e reduzir sua complexidade.

Em seguida, uma etapa de análise de dependência é realizada para extrair relacionamentos gramaticais entre os elementos da frase. Essas informações são encaminhadas para um processador denominado semântico, que orienta a tradução do inglês para uma representação interna intermediária. Deve-se observar que tal módulo depende de um conjunto de regras elaboradas à mão, sendo inevitavelmente específicas do domínio e só podem funcionar em cenários restritos. Finalmente, a representação intermediária pode ser convertida em vários formalismos, incluindo LTL.

Em Sturla e Giancarlo (2017), um algoritmo é apresentado para traduzir uma propriedade especificada usando um subconjunto predefinido do inglês (referido para inglês controlado), para LTL. A abordagem se baseia apenas em propriedades sintáticas, baseada em técnicas de processamento de texto como análise gramatical de dependência.

O autor Chen (2018) apresenta um meio de mapear sentenças em inglês para propriedades LTL. Para restringir o problema, o idioma de destino foi restrito apenas ao subconjunto de LTL, e a linguagem de entrada foi restrita a um subconjunto controlado do inglês. Para os propósitos do projeto, o trabalho focou em sistemas sequenciais de tomada de decisão sintetizados a partir do *Planning Domain Definition Language (PDDL)* que é uma linguagem de planejamento de IA padronizada.

Para a tradução foi utilizado um formalismo gramatical lexicalizado chamado CCG (livre de supercontexto) que define sistematicamente como as palavras se combinam para formar frases e sentenças. Uma das dificuldades encontradas, foi que muitos dos domínios PDDL são mal codificados e não são passiveis de tradução fácil.

Além disso, outra dificuldade foi trabalhar com predicados representados por palavras hifenizadas como : *"communicate_soil_data(X)´´* e *"communicate_rock_data(X)´´* ao invés de *"communicate(X, Y)´´*.

Os autores Wang et al. (2021) apresentaram um trabalho focado no domínio de comandos com aspecto temporal, cuja semântica foi capturada através da Lógica Temporal Linear, LTL. Nesse trabalho é apresentado um analisador que é treinado com pares de sentenças. No momento do treinamento, o analisador apresenta a hipótese de uma representação para a entrada como uma fórmula em LTL.

Os autores utilizaram uma tradução baseada em redes neurais para traduzir uma linguagem natural, o inglês para LTL. A rede utilizada para a tradução foi uma rede LSTM com arquitetura codificador-decodificador. O analisador conseguiu descobrir a estrutura de uma linguagem de entrada e produzir fórmulas executáveis para comandar um robô.

Em Maziero, Pardo e Nunes (2016) foi abordado a RST (*Rhetorical Structure Theory*), que consiste em uma teoria discursiva. Nesse estudo foi apresentada uma segmentação incorporado ao sistema DiZer (*DIscourse analyZER for BRazilian Portuguese*), onde foi substituindo a etapa de segmentação textual, com o objetivo de melhorar o seu desempenho. O método apresentado baseia-se em informações morfossintáticas produzidas pelo parser PALAVRAS. O processo apresentado se mostrou eficiente para o conhecimento morfossintático.

Em Alves (2019) é apresentado um estudo para avaliar e detectar notícias falsas escritas em língua portuguesa através das redes recorrentes do tipo Long Short-Term Memory(LSTM). Duas bases de dados foram utilizados para a realização de experimentos com 36 configurações diferentes. O estudo apresentou boa capacidade

discriminativa na tarefa de classificação das notícias falsas.

A Tabela 2.6 apresenta uma comparação com as principais características dos trabalhos considerados.

Tabela 2.6 – Trabalhos Relacionados

Trabalho	Ling Entrada	Ling Saída	Abordagem
1 Nelken e Francez (1996)	LN Inglês	ACTL	Regras
2 Dwyer, Avrunin e Corbett (1999)	LN Inglês	LTL	Regras
3 Dzifcak et al. (2009)	LN Inglês	CTL	Regras
4 Raman et al. (2013)	LN Inglês	LTL	Regras
5 Yan, Cheng e Chai (2015)	LN Inglês	LTL	Regras
6 Harris e Harris (2015)	LN Inglês	CTL	R. Neurais
7 Ghosh et al. (2016)	LN Inglês	Vários	Regras
8 Sturla e Giancarlo (2017)	LN Inglês	LTL	Regras
9 Chen (2018)	LN Inglês	LTL	Regras
10 Wang et al. (2021)	LN Inglês	LTL	R. Neurais
11 Maziero, Pardo e Nunes (2016)	LN Português	-	Regras
12 Alves (2019)	LN Português	-	R. Neurais

Fonte: Elaborado pelo Autor.

Através dos estudos levantados, é possível afirmar que a maioria dos trabalhos desenvolvidos na área de tradução de propriedades, realiza o processamento de linguagem natural inglesa. Esse trabalho aborda a tradução do português para lógica temporal, além disso, os trabalhos encontrados focam em um categoria de abordagem para a tradução: baseada em regras ou baseada em rede neural. Este trabalho apresenta às duas abordagens e através dos resultados obtidos faz um comparativo sobre elas.

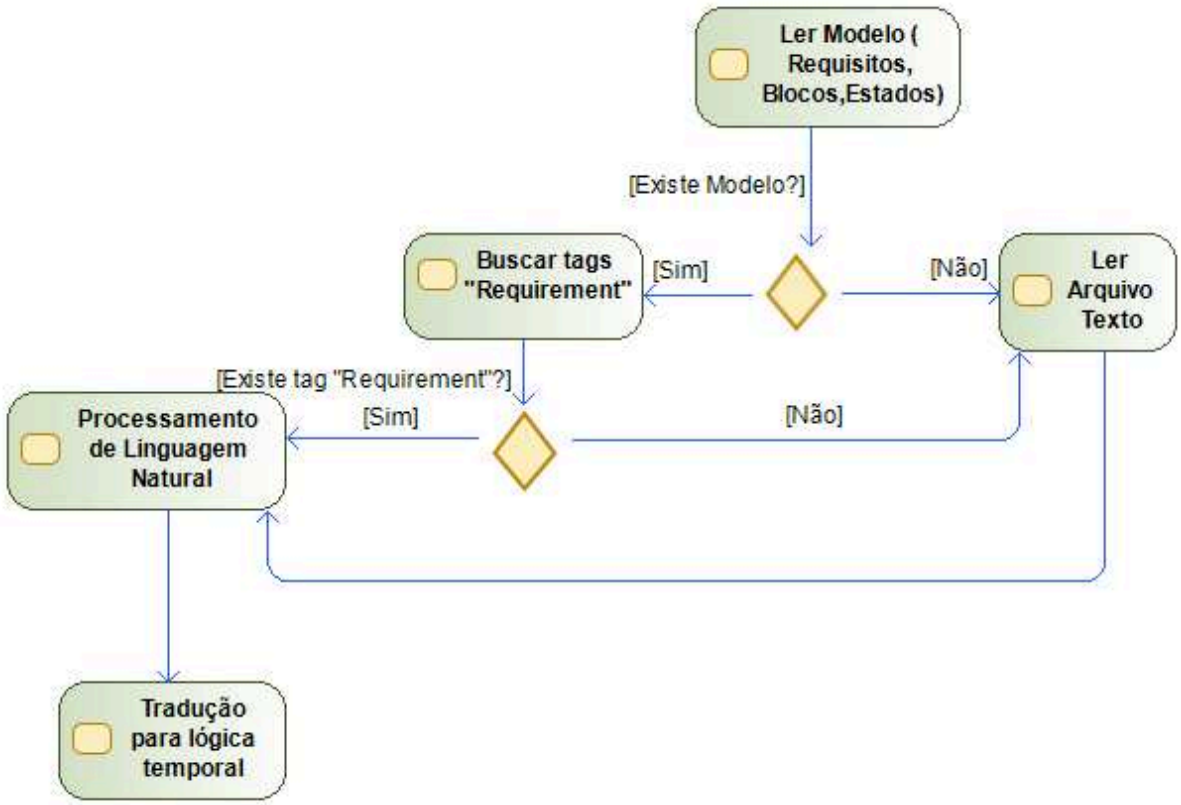
2.10 CONSIDERAÇÕES FINAIS

O Capítulo apresentou as definições utilizadas nesse trabalho. Nesse capítulo a ferramenta de verificação foi escolhida segundo suas características de modelagem e linguagens de propriedades. Além das definições e termos técnicos explanados, foram apresentados estudos que contribuíram o entendimento das propriedades temporais e dos métodos de processamento da linguagem natural.

3 TRADUÇÃO DE REQUISITOS EM PROPRIEDADES TEMPORAIS

Esse Capítulo apresenta a abordagem que será utilizada no desenvolvimento desse trabalho. A ideia geral da abordagem pode ser resumida na Figura 3.1.

Figura 3.1 – Abordagem Tradução de Propriedades



Fonte: Elaborado pelo Autor.

Seguindo os princípios de Baier e Katoen (2008), o processo de verificação do modelo irá analisar as propriedades geradas, para identificar se elas foram satisfeitas ou não, caso não seja, o verificador de modelos irá gerar um contra exemplo e identificar a falha no modelo.

Para que a verificação seja realizada será necessário que os modelos possuam requisitos verificáveis. Os requisitos deverão ser escritos em Português. Durante o processo de tradução, as informações contidas nas *tags* de requisitos ou em um arquivo texto serão analisadas através do processamento de linguagem natural.

Essa abordagem visa traduzir os requisitos do modelo escrito em Português para propriedades formais.

As seções seguintes, mostram mais detalhadamente o procedimento abordado.

3.1 GERAÇÃO DE PROPRIEDADES TEMPORAIS LINEARES USANDO PROCESSAMENTO LINGUAGEM NATURAL

A geração de propriedades é uma das etapas fundamentais na composição do objetivo desse trabalho, pois o verificador de modelos irá realizar a verificação através da análise das propriedades temporais.

Conforme mencionado no início desse Capítulo, para que as propriedades sejam geradas é necessário reconhecer e aplicar o processamento de linguagem natural nos requisitos do modelo. Para realizar o processamento e linguagem natural, os seguintes pontos foram definidos:

1. Definição da linguagem de programação

   Para construir o algoritmo de aprendizagem profunda, foi escolhida a linguagem de programação *Python* em sua versão 3.7. A escolha dessa linguagem foi feita devido a disponibilidade de bibliotecas de suporte aos algoritmos de aprendizagem profunda (*deep learning*) como, por exemplo, a biblioteca *Tensorflow* (ABADI et al., 2015).

   A biblioteca *Tensorflow* foi desenvolvida pela equipe do Google Brain em 2015, que possui estruturas prontas para aplicação de treinamento de redes neurais. Entre as arquiteturas suportadas, está a arquitetura de RNNs com mecanismo de atenção. Essa arquitetura permite a tradução de sentenças de uma linguagem para outra (ABADI et al., 2015).

   Adicionalmente, para processamento com a abordagem baseado em regras, existe a biblioteca *stanza* [1], escolhida para ser utilizada nesse contexto. Essa biblioteca pode ser utilizada para converter uma *string* contendo um texto em linguagem natural em listas de frases e palavras, além disso é capaz de gerar formas básicas das palavras, suas classes gramaticais e características morfológicas, para fornecer uma análise de dependência de estrutura sintática.

2. Definição dos objetivos

   O algoritmo construído deve atender aos seguintes critérios de projeto:

   - deve conseguir ler um arquivo de modelo contendo as definições dos seguintes modelos: Diagrama de Classes (Blocos), Diagrama de Requisitos e Diagrama de Estados.
   - deve conseguir ler os requisitos contidos no diagrama de requisito.
   - se não existir o diagrama de requisito, deve ler os requisitos de um arquivo texto.

[1] Stanza: <stanfordnlp.github.io/stanza/>

- deve conseguir reconhecer um requisito escrito em Português.
- deve conseguir identificar os processos nos elementos do diagrama de classes no arquivo de modelo.
- deve conseguir identificar os estados nos elementos do diagrama de estados no arquivo de modelo.
- deve salvar os processos e os estados em um arquivo de mapeamento.
- deve traduzir a sentença para lógica temporal CTL usada pelo UPPAAL.
- deve escrever um arquivo com extensão ".q" contendo as lógicas temporais traduzidas.

A seguir serão apresentadas as abordagens utilizadas para implementação do algoritmo.

3.1.1 Geração de propriedades temporais lineares usando tradução baseada em redes neurais

Conforme discutido no Capítulo 2, a abordagem de aprendizado profundo é uma das formas utilizadas para processamento de linguagem natural. Nesse trabalho essa abordagem foi aplicada na tradução de linguagem natural para lógicas temporais.

A abordagem com aprendizado profundo consiste nas seguintes etapas:

1. Definição da abordagem de aprendizado profundo a ser utilizada;
2. Definição de um conjunto de dados de entrada do modelo;
3. Treinamento com aprendizado profundo utilizando *Tensorflow*.

Considerando a etapa 1, uma vez definidos os requisitos do sistema, é possível observar que o objetivo principal é a realização da tradução automática de uma determinada linguagem para outra. Mais especificamente objetiva-se traduzir linguagem natural em lógica temporal.

Considerando esse cenário, a abordagem se concentra na tarefa de Tradução Automática Neural (TAN), utilizando um modelo de Rede Neural Recorrente (RNN) do tipo LSTM com arquitetura codificador decodificador, e uma rede *transformer*.

Seguindo para a etapa 2 que se consiste na definição de um conjunto de dados de entrada do modelo. Para realizar a tradução com aprendizagem profunda, é necessário definir uma base de dados para realização do treino do algoritmo.

A etapa 3 consiste no ajuste da rede utilizada na abordagem de tradução. Esses ajustes serão realizados conforme os parâmetros definidos em cada rede. Nas seções abaixo serão apresentadas as arquiteturas de rede neural utilizadas nesse trabalho e a etapa de preparação dos dados.

3.1.1.1 Preparação de Dados

Antes de iniciar o treinamento das redes, é necessário carregar um conjunto de dados. Esse conjunto de dados contém sentenças em português e suas respectivas correspondências em lógica temporal CTL-UPPAAL.

Após o carregamento do conjunto de dados é necessário realizar o pré processamento dessas informações. O pré processamento consiste na limpeza dos dados e na adição de identificadores, em outras palavras, adicionar identificação de início e fim para cada frase e limpar as frases removendo caracteres especiais que não são considerados relevantes nas sentenças.

Como a linguagem de entrada considerada para esse trabalho é o português, os acentos das palavras foram mantidos. Além disso, para a linguagem alvo, alguns caracteres especiais foram mantidos, visto que esses caracteres possuem um significado semântico para a propriedade. Exemplos desses caracteres são: parenteses (“(”,“)”), símbolo de menos (“-”), símbolo de mais (“+”), símbolo de maior (“>”), entre outros.

Após a limpeza dos dados, é realizado um sequenciamento das sentenças, ou seja, cada sentença é transformada em um vetor de palavras como mostra o exemplo da Tabela 3.1.

Tabela 3.1 – Sentenças convertidas em vetores de palavras

Sentenças	Propriedades
“Se”, “o”, “assunto”, “não”, “estiver”, “na”, “lista”, “,”, “o”, “livro”, “não”, “pode”, “ser”, “encontrado”	“(”, “assunto”,“_”,“na”,“_”,“lista”, “=”, “0”,“)”, “-”,“>”, “!”,“(”,“livro”,“_”,“encontrado”, “=”, “1”,“)”
“Se”, “o” ,“número”, “não”, “estiver”, “na” ,“lista”, “,” ,“o” ,“livro” ,“não”, “pode”, “ser”, “encontrado”	“(”,“número”,“_”,“na”,“_”,“lista”, “=”, “0”,“) ”,“-”,“>”, “!”,“(”,“livro”,“_”,“encontrado”, “=”, “1”,“)”,“)”
“Se”, “o”, “usuário”, “inserir”, “os”, “dados”, “corretos”, “,”, “o”, “botão”, “OK”, “é”, “habilitado”	“(”,“usuário”,“_”,“inserir”,“_”,“corretos”, “-”,“>”, “(”,“botão”,“_”,“OK”, “=”, “habilitado”,“)”,“)”
“O”, “Filósofo”, “1”, “e”, “o”, “Filósofo”, “2”, “sempre”, “comem”	“A”,“[”,“]”, “(”,“Filosofo1”,“.”,“comer”, “and”, “Filosofo2”,“.”,“comer”,“)”

Fonte: Elaborado pelo Autor.

Após a etapa de sequenciação são adicionados identificadores a cada palavra da base. Essa etapa transforma os vetores de palavras em vetores numéricos. No exemplo da Tabela 3.2, é possível visualizar as sentenças da Tabela 3.1 após a adição dos identificadores.

Tabela 3.2 – Sentenças convertidas em vetores de identificadores

Sentenças	Propriedades
1, 2, 3, 4, 5, 6, 7, 8, 2, 9, 4, 10, 11, 12	1, 2, 3, 4, 3, 5, 6, 7, 8, 9, 10, 11, 1, 12, 3, 13, 6, 14, 8
1, 2, 13, 4, 5, 6, 7, 8, 2, 9, 4, 10, 11, 12	1, 15, 3, 4, 3, 5, 6, 7, 8, 9, 10, 11, 1, 12, 3, 13, 6, 14, 8, 8
1, 2, 14, 15, 16, 17, 18, 8, 2, 19, 20, 21, 22	1, 16, 3, 17, 3, 18, 9, 10, 1, 19, 3, 20, 6, 21, 8, 8
2, 22, 23, 24, 2, 22, 25, 26, 27	27, 28, 29, 1, 22, 23, 24, 25, 26, 23, 24, 8

Fonte: Elaborado pelo Autor.

No algoritmo é necessário definir o tamanho dos vetores das sentenças. Vetores muito longos podem ser reduzidos em um tamanho *x*, e vetores curtos, menores que o valor definido por *x*, são complementados com zeros. Esses vetores são complementados utilizando *"Padding"*. A Tabela 3.3 apresenta o resultado desse processamento aplicado a Tabela 3.2 com $x = 14$ para as sentenças de entrada, e $x = 20$ para as propriedades.

Tabela 3.3 – Padding aplicado as sentenças

Sentenças	Propriedades
1, 2, 3, 4, 5, 6, 7, 8, 2, 9, 4, 10, 11, 12	1, 2, 3, 4, 3, 5, 6, 7, 8, 9, 10, 11, 1, 12, 3, 13, 6, 14, 8, 0
1, 2, 13, 4, 5, 6, 7, 8, 2, 9, 4, 10, 11, 12	1, 15, 3, 4, 3, 5, 6, 7, 8, 9, 10, 11, 1, 12, 3, 13, 6, 14, 8, 8
1, 2, 14, 15, 16, 17, 18, 8, 2, 19, 20, 21, 22, 0	1, 16, 3, 17, 3, 18, 9, 10, 1, 19, 3, 20, 6, 21, 8, 8, 0, 0, 0, 0
2, 22, 23, 24, 2, 22, 25, 26, 27, 0, 0, 0, 0, 0	27, 28, 29, 1, 22, 23, 24, 25, 26, 23, 24, 8, 0, 0, 0, 0, 0, 0, 0, 0

Fonte: Elaborado pelo Autor.

Após essa preparação de dados é possível realizar os treinos utilizando as redes apresentadas nas próximas seções.

3.1.1.2 Modelo decodificador codificador LSTM

Os dados processados são utilizados para alimentar a rede que consiste em um codificador para ler e codificar as sentenças escritas na linguagem de origem e um decodificador para gerar as propriedades na linguagem alvo. Para a construção do codificador-decodificador LSTM, foram utilizadas 7 camadas agrupadas da seguinte forma:

a) Codificador: para o codificador foram utilizadas a camada de *Input*, a camada *Embedding* e a camada *LSTM*. A quantidade de neurônios da camada *LSTM*

foi definida através do parâmetro *num_units*, e a função de ativação utilizada foi a função *ReLU*.

b) Camada intermediária: Foi utilizada uma camada intermediária *RepeatVector* para ajustar a saída do codificador as necessidades do decodificador visto que a saída do codificador é um vetor de 2 dimensões, já a entrada do decodificador é um vetor de 3 dimensões.

c) Decodificador: o decodificador foi construído com 2 camadas *LSTM* onde a quantidade de neurônios utilizados, foi definida pelo parâmetro *num_units* e a função de ativação utilizada foi a função *ReLU*. Além das duas camadas *LSTM* foi utilizada uma camada *Dense* que implementa saída do modelo.

As camadas são então empilhadas criando o modelo. O codificador faz a codificação da entrada e retorna os resultados para o decodificador, o decodificador retorna os dados decodificados e então são realizados os cálculos de perda e otimização. A função de otimização utilizada pela rede foi a função *Adam*, e como métrica de avaliação do treino, foi utilizada a acurácia que compara a saída obtida com a saída desejada.

Para realização do treino, o usuário pode definir as épocas de treinamento que serão utilizadas. Outro parâmetro que pode ser configurado é a quantidade de neurônios das camadas de LSTM.

3.1.1.3 Transformer

A estrutura da rede *transformer* pode ser observada na Figura 2.11. Essa rede possui varias etapas de processamento e também é dividida em camadas. O processamento e as camadas serão explanadas a seguir:

a) Camada *Embedding*: As camadas de *Input Embedding* e *Output Embedding* são nativas do próprio tensorflow. Essas camadas recebem como entada a base de dados onde o Output Embrdding recebe os dados na linguagem de destino e o Input Embedding recebe os dados na linguagem de origem. A saída da função embedding é enviada para o *Positional Encoding*.

   Por não usar a recorrência e convolução, é necessário adicionar informações sobe a posição absoluta e relativa dos identificadores em uma sequência. Esse processo é chamado de *Positional Encoding*.

b) Na camada do *Positional Encoding* são utilizadas duas equações, A Equação 3.1 aplicada as palavras pares e a Equação 3.2 para palavras impares (VASWANI et al., 2017):

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{3.1}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{3.2}$$

c) Camada Mecanismo de atenção: O *transformer* possui um mecanismo de atenção chamado *Multi-head Attention*. O *Scaled Dot-Product Attention* é uma camada interna do *Multi-head Attention* sendo definido pela Equação 3.3 (VASWANI et al., 2017).

$$Attention(Q, K, V) = softmax(QK^T/\sqrt{d_k})V \tag{3.3}$$

Na Equação 3.3, Q, K e V são matrizes que representam sentenças depois da etapa do Embedding. Através desse mecanismo a rede consegue entender o grau de ligação entre as palavras contidas na sentença.

Quanto mais similaridade existir entre as palavras, o resultado da similaridade retorna mais próximo do valor 1, caso o valor de similaridade seja 0, não existe correlação entre as palavras. Quanto mais próximo do resultado -1, significa que a correlação é inversa. Esse conceito é implementado tanto para o codificador quanto para o decodificador separadamente, e recebe o nome de *self-attention*.

O mecanismo de atenção interligado às duas camadas: codificador e decodificador, é chamada de *encoder-decoder attention* e esse mecanismo verifica a similaridade entre as palavras da sentença de entrada e das palavras da sentença de saída. Dessa forma, é possível calcular os índices de similaridades, ou seja, como as sentenças da linguagem de entrada se relacionam com as sentenças de linguagem de saída.

d) Camada *Rede feed-forward*: O Codificador e decodificador contém uma rede feed-forward totalmente conectada. A rede consiste em duas transformações lineares com uma ativação *ReLU* no meio.

e) Camadas Codificador e Decodificador: O codificador é composto de duas sub-camadas: A *Multi-head Attention*, e a rede *feed forward*.

Essas camadas são seguidas pelo processo de adição e normalização que é responsável por manter os dados na mesma escala, além disso, ela auxilia a rede para que os dados anteriores não sejam esquecidos, guardando um *backup* dos dados antes do próximo processamento. A segunda camada é a *Rede feed-forward* totalmente conectada. A rede consiste em duas transformações lineares com uma ativação ReLU entre elas.

Já o Decodificador possui uma camada a mais, interligada com a saída do codificador. Essa camada é a *encoder-decoder attention*.

Para o treinamento os parâmetros d_model que define a quantidade de colunas da matriz de *Embbeding*, o parâmetro nb_layers que define a quantidade de codificadores utilizados, e o parâmetro ffn_units que são as camadas da rede feed-forward, podem ser ajustados. Além desses parâmetros a quantidade de épocas também pode ser ajustada.

3.1.1.4 Validação dos resultados

Para avaliar os resultados obtidos, foi utilizada a metodologia de validação cruzada. Esse método utiliza o conceito de tradução supervisionada, onde é passado para a rede um conjunto de dados contendo valores de entrada e valores de saída, no caso do tradutor as sentenças escritas na linguagem de origem e as correspondências dessas sentenças escritas na linguagem alvo.

O conjunto de dados é dividido em *n* partes, por exemplo, se n = 3 então duas partes são utilizadas para treino e uma parte é utilizada para avaliação. Esse processo é realizado até que todas as partes sejam usadas para treino e validação (KOHAVI, 2001).

3.1.2 Geração de propriedades temporais lineares usando tradução baseada em regras

O tradutor baseado em regras é uma das abordagens utilizadas no processamento de linguagem natural. Para construção do tradutor baseado em regras no contexto de tradução de sentenças foram realizadas as seguintes etapas:

- Definição das regras do reconhecedor de sentenças.
- Construção do tradutor baseado em regras.

3.1.2.1 Definição das regras do reconhecedor de sentenças

O tradutor baseado em regras não é capaz de aprender o significado de uma palavra e como essa palavra se relaciona com as outras palavras da sentença, por esse motivo é necessário definir regras que permitam o seu reconhecimento.

Neste trabalho a propriedade temporal pode ser composta pela função denominada processo, o estado que poderá ou não ser alcançado pelo processo, os quantificadores temporais, os operadores lógicos, e as variáveis.

O processo deverá ser definido na sentença como um nome próprio ou como uma palavra antecedida de um determinante (o, a, os, as), por exemplo: "***Filósofo1*** *e* ***Filósofo2*** *sempre comem.*", nessa sentença os processos são: **Filósofo1**, **Filósofo2**. A sentença: "*Se o* ***assunto*** *não estiver na lista, o* ***livro*** *não pode ser encontrado*."

exemplifica quando o processo é uma palavra antecedida de um determinante (o, a, os, as). Neste exemplo os dois processos são: **assunto**, **livro**.

O predicado que sucede o (os) processo (processos), ou seja, a ação, é reconhecido como o estado que o processo poderá acessar em algum momento. No exemplo: "***Filósofo1*** *e o* ***Filósofo2*** *sempre comem.*", o estado reconhecido seria *comer*.

A sentença também poderá expressar que um processo não consegue atingir um certo estado, por exemplo: "*Se o* ***Filósofo1*** *estiver comendo então o Filósofo2 não está comendo*". A ideia de não atingir um certo estado é representada pela palavra "não". Ao escrever uma sentença, os quantificadores temporais: Sempre (A[]), Potencialmente (E<>) ,Eventualmente (A<>), Nunca (A[] *not*), Possivelmente(E[]), devem ser definidos em cada sentença, por exemplo: "*Se Filósofo1 e Filósofo2* ***nunca*** *comem ao mesmo tempo*".

A conexão existente nas propriedades temporais de causa e efeito também devem ser introduzidas na sentença. Para expressar a ideia de que *p* provoca a ocorrência de *q* (*imply*), é necessário adicionar um conector na sentença: *"obriga que","causa","resulta em","se ..então"*. Por exemplo: *Se Filósofo1 está comendo,* ***então*** *Filósofo2 está pensando.*

Para definir uma propriedade onde *p* provoca a ocorrência de *q* em algum momento ($\rightarrow$) é necessário adicionar um conector na sentença: *"futuramente","posteriormente"*. Por exemplo: *Se Filósofo1 está comendo,* ***futuramente*** *Filósofo2 estará pensando.*

Se o objetivo for expressar uma propriedade onde existem comparações com alguma variável ou expressões matemáticas, é necessário adicionar as correspondências dos operadores lógicos e matemáticos definidos na Tabela 3.4, por exemplo: "Se c é **igual** a 1 **e** b é **igual** a 2, então o Filósofo1 está comendo".

Tabela 3.4 – Operadores lógicos e matemáticos

Operador	Palavra
+	mais
and	e
or	ou
==	"igual a", igual
>	"maior que", maior
>	"menor que", menor
>=	"maior ou igual"
<=	"menor ou igual"
!=	"diferente de"

Fonte: Elaborado pelo Autor.

Em síntese a definição dessas regras pode ser expressa como:

```
FÓRMULA = EXPRESSÃO | EXPRESSÃO CONECTOR EXPRESSÃO
```

onde:

```
EXPRESSÃO: PROCESSOS QUANTIFICADOR ESTADO | PROCESSOS ESTADO |
        QUANTIFICADOR PROCESSOS ESTADO | EXPRESSÕES_MATEMÁTICAS |
        QUANTIFICADOR EXPRESSÕES_MATEMÁTICAS

EXPRESSÕES_MATEMÁTICAS : EXPRESSÕES_MATEMÁTICAS e EXPRESSÕES_MATEMÁTICAS |
         EXPRESSÕES_MATEMÁTICAS ou EXPRESSÕES_MATEMÁTICAS |
         EXPRESSÃO_MATEMÁTICA

PROCESSOS = PROCESSO | PROCESSOS e PROCESSOS | PROCESSOS ou PROCESSOS

EXPRESSÃO_MATEMÁTICA = VARIÁVEL OPERADOR_MATEMÁTICO VALOR_NUMÉRICO |
                       VARIÁVEL OPERADOR_MATEMÁTICO VARIÁVEL
```

A próxima seção apresenta como o código foi estruturado.

3.1.2.2 Construção do tradutor baseado em regras

O código do tradutor segue a estrutura apresentada no Algoritmo 3:

Algoritmo 1: Verificar tag Requirement

```
1 if Existe elemento "Requirement" then
2     for "Requirement" em modelo do
3         salva conteúdo de "Requirement" em arquivo texto;
4     end
5 end
```

a) O código possui um arquivo de inicialização. Neste arquivo são definidos o arquivo de entrada, o arquivo de modelo, o arquivo de saída, e os arquivos contendo o dicionário de palavras.

b) Quando existir um arquivo de modelo no arquivo de inicialização, o tradutor realizada uma busca pelo atributo *text* nos elementos *"Requirements"*. Esse atributo é encontrado na seção do arquivo onde são definidas as estruturas correspondentes ao **diagrama de requisitos**, um exemplo desse elemento pode ser observado na Listagem 3.1. O conteúdo é então considerado como a sentença a ser traduzida.

```
<Requirements:Requirement  id="2" text="O filosofo 1 e
  o filosofo 2 sempre comem" />
```

Listagem 3.1 – Elemento Requiremens

c) Caso o elemento *"Requirements"* não exista no arquivo do modelo, o usuário deve inserir os requisitos em um arquivo texto.

Algoritmo 2: Analisar Requisitos

```
1  if Existe modelo then
2      for processo encontrado no Requisito do
3          if Não existe processo no modelo then
4              Pergunta ao usuário o nome do processo;
5          end
6          Salvar processo no arquivo de Mapeamento;
7      end
8      Atualizar os processos na sentença;
9      Priorizar os marcadores temporais;
10     for estados em Requisitos do
11         if Não existe estados no modelo then
12             Pergunta ao usuário nome das estados;
13         end
14         Salvar estados no arquivo de mapeamento;
15     end
16 else
17     for processos em Requisitos do
18         Salvar Processos no arquivo de mapeamento;
19     end
20     for estados em Requisitos do
21         Salvar estados no arquivo de mapeamento;
22     end
23 end
```

Algoritmo 3: Tradutor

```
1  ler arquivo de inicialização;
2  if Existe Modelo then
3      Verificar tag Requirement;
4  end
5  for Requisitos em arquivo texto do
6      Analisar Requisitos;
7      Processar Requisito;
8      Traduzir Requisito para propriedade;
9      Escrever propriedade em arquivo .q;
10 end
```

d) O tradutor lê o arquivo contendo os requisitos e reconhece sentença por sentença, onde cada sentença é finalizada com um "\n".

e) As funções da biblioteca *stanza* são responsáveis por realizar um pré processamento das sentenças. Essa biblioteca funciona como um *parser* onde as palavras da sentença são classificadas de acordo com sua classe gramatical, por exemplo:

- O: DET (determinantes);

- filosofo: NOUN (substantivos comuns);
- 1: NUM (numerais);
- e: CCONJ (conjunção);
- o: DET (determinantes);
- filosofo: NOUN (substantivos comuns);
- 2: NUM (numerais);
- sempre: ADV (advérbio);
- comem: VERB (verbo).

Ao processar a sentença o tradutor faz o reconhecimento das palavras que representam os processos. O reconhecimento é feito de acordo com as regras definidas em 3.1.2.1. Se o arquivo de modelo existir, o tradutor executa uma busca por uma ocorrência da palavra que foi classificada como processo.

O tradutor faz a busca pelo conteúdo do atributo *name* nos elementos *nestedClassifier* onde *xmi:type= "uml:Class"* e em *ownedAttribute*. No modelo, cada classe representa um processo e por isso os atributos são encontrados na seção do arquivo onde são definidas as estruturas correspondentes ao **diagrama de classes** Um exemplo desses elementos pode ser observado na Listagem 3.2.

```
<nestedClassifier xmi:type="uml:Class"
                        name="Filosofo">
<ownedAttribute         name="Filosofo1"
                  visibility="public"
                 aggregation="composite" />
```

Listagem 3.2 – Elementos Processos.

Caso exista mais de uma, ou não exista nenhuma ocorrência compatível com a palavra, o tradutor pergunta ao usuário qual dos resultados encontrados poderá ser atribuído a palavra, por exemplo:

No requisito:*"O **filósofo 1** e o **filósofo 2** sempre comem."*, existem duas sequências de palavras reconhecidos como processos pelo tradutor: **filósofo 1** e **filósofo 2**. A Figura 3.2 exibe o resultado da busca referente a sequência **"filósofo 1"**, como não foi encontrada exatamente essa definição, são exibidos todos os resultados aproximados. O Usuário é responsável por definir qual será a palavra será atribuída ao processo.

Figura 3.2 – Resultado de busca Processos - Jantar dos Filósofos.

Fonte: Elaborado pelo Autor.

f) Após o reconhecimento dos processos, o tradutor armazena as correspondências em um arquivo de mapeamento. O exemplo da Listagem 3.3 mostra como é feita a correspondência nesse arquivo.

```
filosofo 1=Filosofo1
```

Listagem 3.3 – Formato do arquivo de mapeamento do modelo.

No exemplo da Lista 3.3 é possível observar que para sequência de palavras "filósofo 1" foi atribuído a palavra "Filosofo1", ou seja, uma das opções retornadas pelo tradutor. O arquivo de mapeamento é utilizado para armazenar a correspondência entre as palavras identificadas como processo, essa correspondência será utilizada na tradução da próxima sentença que possuir a palavra identificada.

g) Após o mapeamento o tradutor atualiza a sentença com o novo valor atribuído ao processo. No exemplo a sequência de palavra "filósofo 1" foi substituída por: "Filosofo1" e a sentença passa a ter a seguinte construção: "O **Filosofo1** e o **filósofo 2** sempre comem.". O mesmo procedimento é realizado para encontrar a correspondência para a sequência de palavras **filósofo 2**.

h) O próximo passo que o tradutor realiza é transformar os quantificadores em *tags* identificando o começo e o fim da sub-sentença referente a esse quantificador. Para o exemplo, a sentença é substituída por: <sempre> O **Filosofo1** e o **Filosofo2** comem </sempre>

i) Os marcadores temporais podem ser modificados no arquivo do dicionário respeitando as seguintes classes:

   - A<>: Eventualmente
   - A[]: Sempre
   - E<>: Possivelmente
   - E[]: Potencialmente

j) Os conectores também podem ser modificados respeitando a seguinte classificação: depois, implica. O dicionário é definido como um arquivo JSON[2] com a estrutura exibida na Listagem 3.4.

[2] JavaScript Object Notation: <www.json.org>

```
{
    "palavra": "posteriormente",
      "lemma": "depois",
       "upos": "ADV"
}
```

Listagem 3.4 – Formato do arquivo de json.

A chave “lemma” define a classe de agrupamento, nesse caso: “depois”.

k) Para definir os estados, o tradutor verifica a sequência de palavras que definem uma ação, ou seja, o predicado da sentença. O estado é construído até que um quantificador temporal, um conector, operador lógico ou matemático, ou um novo processo sejam encontrados.

Caso o arquivo de modelo exista, é realizada uma busca no arquivo pela ocorrência da sequência definida como estado. No modelo, os estados são encontrados na seção onde estão definidas as estruturas do **diagrama de estado**. A busca é realizada nos elementos *<subvertex>* onde *xmi:type="uml:State"*, de acordo com a Listagem 3.5.

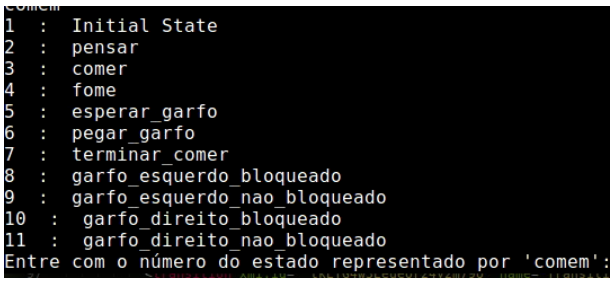
```
<subvertex xmi:type="uml:State"  name="comer"/>
```

Listagem 3.5 – Elemento subvertex.

Caso exista mais de uma, ou não exista nenhuma ocorrência compatível com a sequência palavras que represente o estado, o tradutor pergunta ao usuário qual dos resultados encontrados poderá ser atribuído ao estado, por exemplo, em: <sempre> O Filosofo1 e o Filosofo2 **comem** </sempre>. A palavra reconhecida pelo tradutor como predicado nessa sentença é: **comem**.

A Figura 3.3 exibe o resultado de busca referente a palavra **“comem”**. Como não foi encontrada exatamente essa palavra, são exibidos todos os resultados aproximados. O Usuário é responsável por definir qual será a palavra correspondente ao estado.

Figura 3.3 – Resultado de busca: Ação - Jantar dos Filósofos



Fonte: Elaborado pelo Autor.

l) Após o reconhecimento dos estados, o tradutor armazena as correspondências no arquivo de mapeamento, conforme mostrado na Listagem 3.6.

```
comem=comer
```

Listagem 3.6 – Estados - arquivo de mapeamento.

O arquivo de mapeamento é utilizado para armazenar a correspondência entre as palavras identificadas como estado, essa correspondência será utilizada na tradução da próxima sentença que possuir a palavra identificada.

m) Após o mapeamento o tradutor atualiza a sentença com o novo valor atribuído ao estado. No exemplo a palavra “comem” foi substituída por: “comer” e a sentença passa a ter a seguinte construção: “<sempre> O Filosofo1 e o Filosofo2 **comer** </sempre>”.

n) O próximo passo do tradutor é remover palavras que não serão usadas na construção da propriedade, como o determinante “O”. No exemplo, excluindo esse determinante a sentença fica da seguinte forma : “<sempre> Filosofo1 e Filosofo2 **comer** </sempre> ”.

o) Caso o predicado identificado como estado seja referente a mais de um processo como no exemplo: “<sempre> Filosofo1 e Filosofo2 **comer** </sempre> ”, o tradutor atribuirá esse estado aos respectivos processos. A sentença será atualizada para:“ <sempre> Filosofo1.**comer** e Filosofo2.**comer**</sempre>”.

p) Os quantificadores temporais, conectores e operadores são substituídos pela palavra reservada correspondente, e finalmente a sentença é gravada como uma propriedade temporal em um arquivo com extensão .q (aceito pelo verificador UPPAAL). Como resultado de tradução do exemplo, o tradutor retorna a seguinte propriedade: “A[] (Filosofo1.**comer** and Filosofo2.**comer**)”

q) Caso não exista um arquivo de modelo contendo os diagramas de estado e diagrama de classe, todo o processo seria realizado, no entanto, o tradutor utilizaria as próprias palavras (ou sequências de palavras) que foram identificadas como processo e estado na tradução da propriedade. O exemplo seria construído da seguinte forma: “A[] (filosofo_1.**comem** and filosofo_2.**comem**)”

r) O tradutor não possui tratamento de erro, logo qualquer sentença escrita fora das regras determinadas em 3.1.2.1 será traduzida de forma errônea pelo tradutor.

3.2 CONSIDERAÇÕES FINAIS

Este Capítulo apresentou as abordagens utilizadas para tradução de propriedades através do processamento de linguagem natural. Além disso, foram apresentadas características importantes sobre os algoritmos utilizados. No próximo Capítulo, serão apresentados os experimentos realizados e os resultados desses experimentos.

4 RESULTADOS E DISCUSSÕES

As abordagens de tradução baseada em regras e tradução neural automática foram aplicados em três casos de estudo para testar a tradução de propriedades em linguagem natural em lógica temporal. Nas seções seguintes serão apresentados as etapas do processo de tradução das especificações e os resultados obtidos.

4.1 CASO DE ESTUDO: JANTAR DOS FILÓSOFOS

O problema do jantar dos filósofos é um dos problemas clássicos de sincronização e pode ser ilustrado da seguinte maneira (RAMADHAN; SIAHAAN, 2016):

Há cinco filósofos sentados ao redor de uma mesa como na Figura 4.1. Existem cinco tigelas de macarrão na frente de cada filósofo e um garfo entre cada filósofo, ou seja, 5 garfos.

Os filósofos passam algum tempo pensando (quando estão cheios) e comendo (quando estão com fome). Quando estiver com fome, o filósofo pegará dois garfos, um na mão esquerda e um na mão direita, e então comerá. No entanto, às vezes, não é possível pegar os dois garfos. Se um filósofo conseguir pegar os dois garfos, então os próximos dois filósofos, um da sua esquerda e um da sua direita, devem esperar até que os garfos sejam colocados de volta.

Figura 4.1 – Jantar dos Filósofos

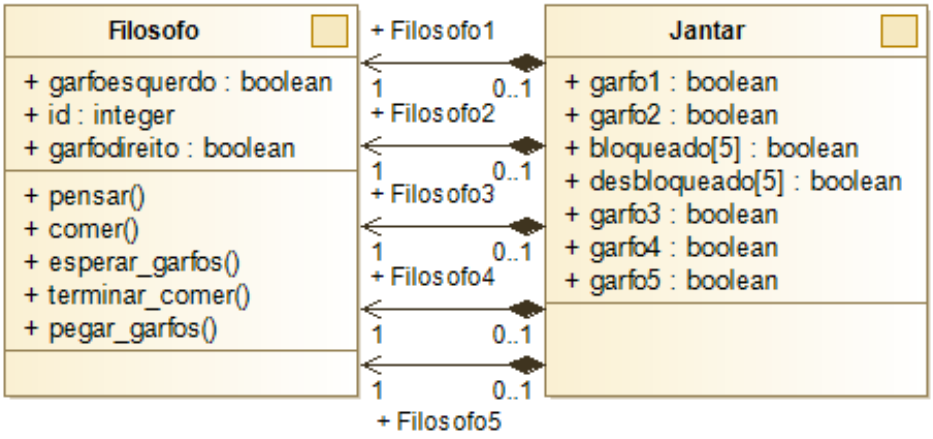
Fonte: Ramadhan e Siahaan (2016, p. 2)

No entanto, o jantar dos filósofos pode apresentar problemas de sincronização. O fato de todos os filósofos executarem o mesmo procedimento, e compartilharem os mesmos recursos, pode causar um *impasse (deadlock) no sistema*. Um filósofo poderia ficar travado esperando um recurso que está sendo utilizado por outro filósofo.

Outro problema é o fato de mais do que um filósofo ficarem com fome em simultâneo, os filósofos famintos tentariam agarrar os garfos em simultâneo. Precisa-se garantir que nenhum filósofo morra de fome.

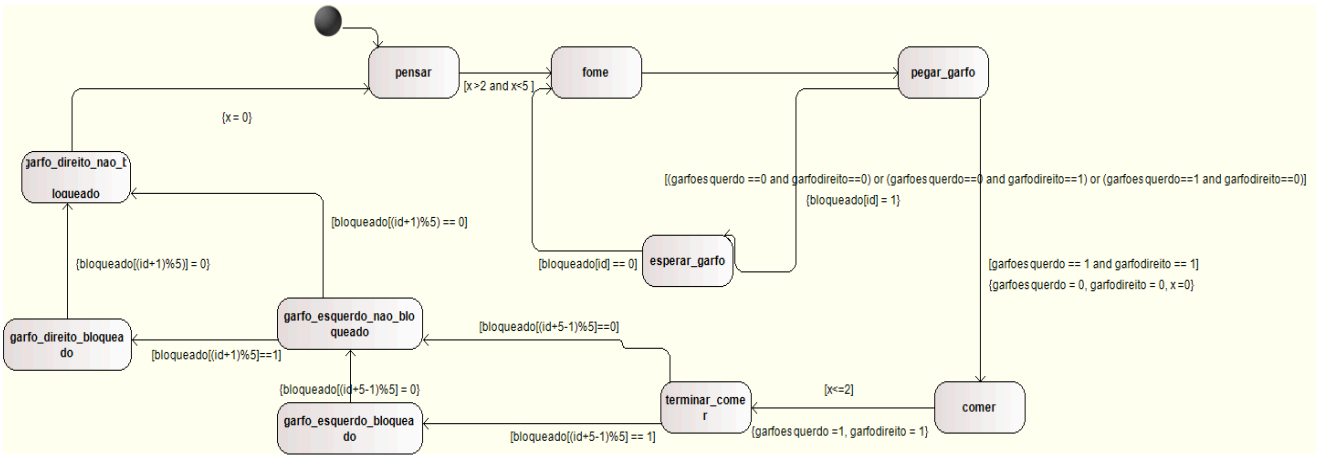
As Figuras 4.2, 4.3 apresentam a modelagem semi-formal em SysML, do problema do Jantar dos Filósofos.

Figura 4.2 – Diagrama de Blocos - Jantar dos Filósofos



Fonte: Elaborado pelo Autor.

Figura 4.3 – Diagrama de Estados - Jantar dos Filósofos



Fonte: Elaborado pelo Autor.

Na Figura 4.3 os comportamentos e estados podem ser descritos da seguinte forma:

- O estado inicial é chamado de “pensar”.
- O próximo estado é chamado de “fome”. Se o filósofo entra no estado de “fome”, ele deverá ir para o próximo estado: “pegar_garfos”.
- No estado “pegar_garfos”, é realizado um processo para verificar se os garfos, da sua direita e da sua esquerda, estão disponíveis. Se estiverem disponíveis, o filósofo pega os garfos e começa a comer.

- Se algum dos garfos estiver ocupado por um dos filósofos vizinhos, o filósofo em questão, irá aguardar os garfos em um estado intermediário, chamado “esperar_garfo”.
- Caso o filósofo consiga pegar os garfos e começar a comer, ele entra no estado chamado “comer” e poderá permanecer lá por no máximo 2 unidades de tempo.
- Após terminar de comer, o filósofo devolve os garfos, e avisa para seus vizinhos, caso estejam esperando os garfos pra comer. Esse processo é realizado utilizando um canal. Após avisar os seus vizinhos, o filósofo entra no estado de pensar.
- O filósofo só poderá tentar pegar os garfos se receber um aviso de que o garfo de seu vizinho está disponível.
- Na modelagem os filósofos foram definidos como: filósofo1, filósofo2, filósofo3, filósofo4, filósofo5

4.1.1 Verificação: Definição das propriedades

Três importantes propriedades são verificadas no exemplo do jantar dos filósofos: ausência de impasse, vivacidade e segurança.

4.1.1.1 Propriedades de Vivacidade

Para que o problema dos filósofos seja solucionado é necessário garantir que todos os filósofos comam em algum momento quando estiverem com fome. Essa propriedade pode ser definida como uma propriedade de vivacidade (*liveness*). De modo mais geral, uma propriedade de vivacidade afirma que “algo bom eventualmente ocorrerá”. A Tabela 4.1 apresenta as propriedades e os requisitos correspondentes escritos em português.

Tabela 4.1 – Propriedades de Vivacidade e Requisitos - Jantar dos Filósofos

Requisito	Propriedade
Se o filósofo 1 estiver com fome, futuramente o filósofo 1 irá comer	Filosofo1.fome $\rightarrow$ Filosofo1.comer
Se o filósofo 2 estiver com fome, futuramente o filósofo 2 irá comer	Filosofo2.fome $\rightarrow$ Filosofo2.comer
Se o filósofo 3 estiver com fome, futuramente o filósofo 3 irá comer	Filosofo3.fome $\rightarrow$ Filosofo3.comer
Se o filósofo 4 estiver com fome, futuramente o filósofo 4 irá comer	Filosofo4.fome $\rightarrow$ Filosofo4.comer
Se o filósofo 5 estiver com fome, futuramente o filósofo 5 irá comer	Filosofo5.fome $\rightarrow$ Filosofo5.comer

Fonte: Elaborado pelo Autor.

4.1.1.2 Propriedades de Ausência de Impasse (deadlock)

A propriedade de ausência de impasse é um complemento da propriedade de segurança, garante que o sistema nunca irá ficar travado esperando por algum recurso que nunca terá. A Tabela 4.2 apresenta a propriedade e o requisito correspondente escrito em português.

Tabela 4.2 – Propriedade de Ausência de Impasse e Requisito - Jantar dos Filósofos

Requisito	Propriedade
Nunca ocorrerá deadlock	$A[]\ not\ deadlock$

Fonte: Elaborado pelo Autor.

4.1.1.3 Propriedades de Segurança

Outra propriedade importante, é a propriedade de segurança. Uma propriedade de segurança afirma que “algo ruim não ocorre”. No problema dos filósofos, se um filósofo estiver comendo, o seu vizinho não poderá comer em simultâneo. Caso essa situação acontecesse, levaria a um *impasse*, ou seja, dois processos teriam acesso simultâneo a um recurso compartilhado. As Tabelas 4.3 e 4.4 apresentam as propriedades e os requisitos correspondentes escritos em português.

Tabela 4.3 – Propriedades de Segurança e Requisitos - Jantar dos Filósofos 1

Requisito	Propriedade
o filosofo 1 e o filosofo 2 nunca comem ao mesmo tempo	A[] not (Filosofo1. comer and Filosofo2.comer)
o filosofo 2 e o filosofo 3 nunca comem ao mesmo tempo	A[] not (Filosofo2. comer and Filosofo3.comer)
o filosofo 3 e o filosofo 4 nunca comem ao mesmo tempo	A[] not (Filosofo3.comer and Filosofo4.comer)
o filosofo 4 e o filosofo 5 nunca comem ao mesmo tempo	A[] not (Filosofo4. comer and Filosofo5.comer)
o filosofo 5 e o filosofo 1 nunca comem ao mesmo tempo	A[] not (Filosofo5. comer and Filosofo1.comer)
sempre que o filosofo 5 estiver comendo implica que o filosofo 4 e o filosofo 1 não estejam comendo	A[] Filosofo5.comer imply ! Filosofo4.comer and ! Filosofo1.comer
sempre que o filosofo 1 estiver comendo implica que o filosofo 5 e o filosofo 2 não estejam comendo	A[] Filosofo1.comer imply ! Filosofo5.comer and ! Filosofo2.comer
sempre que o filosofo 4 estiver comendo implica que o filosofo 3 e o filosofo 5 não estejam comendo	A[] Filosofo4.comer imply ! Filosofo3.comer and ! Filosofo5.comer
sempre que o filosofo 2 estiver comendo implica que o filosofo 3 e o filosofo 1 não estejam comendo	A[] Filosofo2.comer imply ! Filosofo3.comer and ! Filosofo1.comer

Fonte: Elaborado pelo Autor.

Tabela 4.4 – Propriedades de Segurança e Requisitos - Jantar dos Filósofos 2

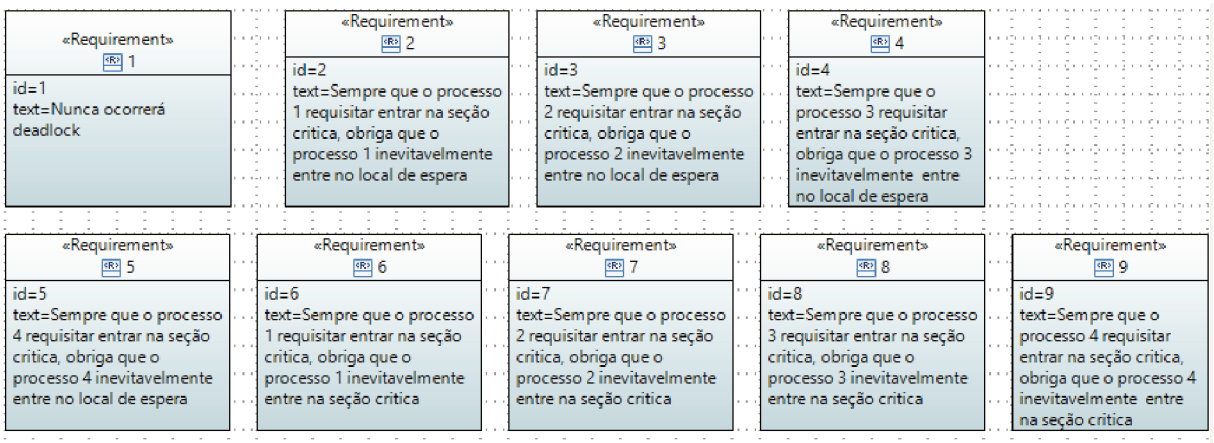
Requisito	Propriedade
sempre que o filosofo 3 estiver comendo implica que o filosofo 4 e o filosofo 2 não estejam comendo	A[] Filosofo3.comer imply ! Filosofo4.comer and ! Filosofo2.comer

Fonte: Elaborado pelo Autor.

4.2 CASO DE ESTUDO: PROTOCOLO DE FISCHER

O protocolo de Fischer é um protocolo de exclusão mútua envolvendo *n* processos, que considera leituras e gravações atômicas em uma variável compartilhada. A exclusão mútua no protocolo de Fischer é garantida colocando limites nos tempos de execução das instruções (VEREIJKEN, 1996). Os diagramas do protocolo de Fischer são definidos nas figuras 4.4, 4.5 e 4.6.

Figura 4.4 – Diagrama de Requisitos - Protocolo de Fischer



Fonte:Elaborado pelo Autor.

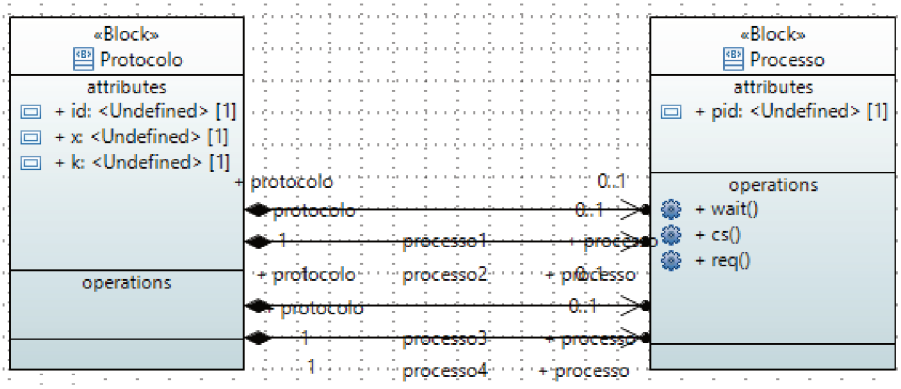
Segundo Behrmann, David e Larsen (2004), o protocolo pode ser descrito da seguinte forma: a partir da localização inicial, se $id == 0$, os processos vão para um estado de requisição, req.

Os processos permanecem no estado req por um período entre 0 e k unidades de tempo, em seguida, vão para o estado $espera$ e definem o id com valor do seu ID de processo (*pid*). Nesse estado, ele deve aguardar pelo menos k unidades de tempo, $x > k$, onde k é uma constante (2), antes de entrar na seção crítica CS se for sua vez, $id == pid$.

O protocolo é baseado no fato de que após k unidades de tempo e se id é diferente de 0, todos os processos que desejam entrar na seção crítica estão esperando para entrar na seção crítica também, mas apenas um tem o ID correto.

Ao sair da seção crítica, os processos são reiniciados para permitir que outros processos entrem na seção crítica CS. Se os processos estão esperando, eles

Figura 4.5 – Diagrama de Bloco - Protocolo de Fischer



Fonte: Elaborado pelo Autor

podem tentar novamente quando o outro processo sair do CS, retornando para req (BEHRMANN; DAVID; LARSEN, 2004).

Foram gerados requisitos de ausência de impasse, e vivacidade para os 4 processos conforme as propriedades definidas em Behrmann, David e Larsen (2004). Essas propriedades são apresentadas nas Tabelas 4.5 e 4.6

Tabela 4.5 – Propriedades e Requisitos - Protocolo de Fischer 1

Requisito	Propriedade
Nunca ocorrerá deadlock	A[] not (deadlock)
Sempre que o processo 1 requisitar entrar na seção critica, obriga que o processo 1 inevitavelmente entre no local de espera	processo1.req –> processo1.espera
Sempre que o processo 1 requisitar entrar na seção critica, obriga que o processo 1 inevitavelmente entre na seção crítica	processo1.req –> processo1.regiao_critica
Sempre que o processo 2 requisitar entrar na seção crítica, obriga que o processo 2 inevitavelmente entre no local de espera	processo2.req –> processo2.espera
Sempre que o processo 3 requisitar entrar na seção crítica, obriga que o processo 3 inevitavelmente entre no local de espera	processo3.req –> processo3.espera
Sempre que o processo 4 requisitar entrar na seção crítica, obriga que o processo 4 inevitavelmente entre no local de espera	processo4.req –> processo4.espera

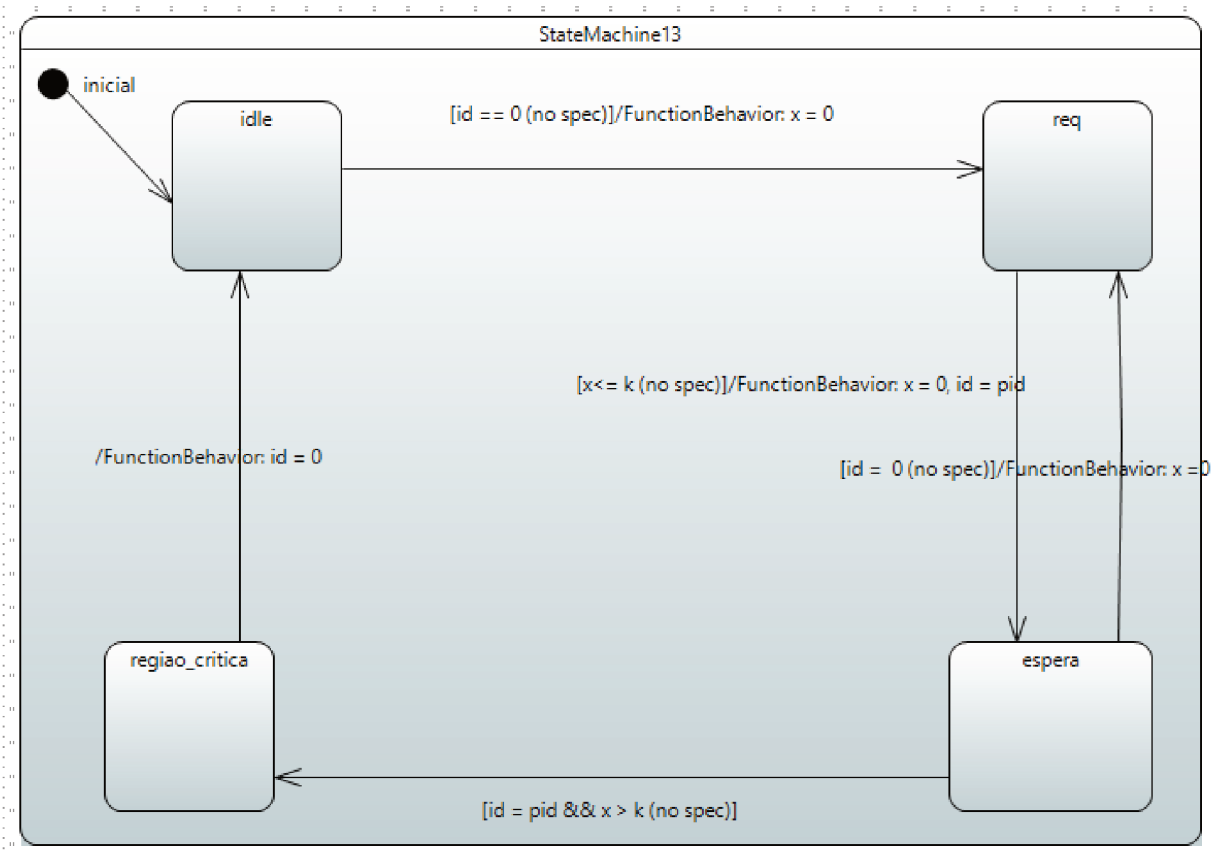
Fonte: Elaborado pelo Autor.

Tabela 4.6 – Propriedades e Requisitos - Protocolo de Fischer 2

Requisito	Propriedade
Sempre que o processo 4 requisitar entrar na seção crítica, obriga que o processo 4 inevitavelmente entre na seção crítica	processo4.req -> processo4.regiao_critica
Sempre que o processo 3 requisitar entrar na seção crítica, obriga que o processo 3 inevitavelmente entre na seção crítica	processo3.req -> processo3.regiao_critica
Sempre que o processo 2 requisitar entrar na seção crítica, obriga que o processo 2 inevitavelmente entre na seção crítica	processo2.req -> processo2.regiao_critica

Fonte: Elaborado pelo Autor.

Figura 4.6 – Diagrama de Estados - Protocolo de Fischer



Fonte: Elaborado pelo Autor

4.3 CASO DE ESTUDO: PADRÕES DE ESPECIFICAÇÃO TEMPORAL DE DWYER

O repositório “Padrões de Especificação Temporal”, coleta informações sobre a especificação de propriedade para verificação de sistemas de estado finito. A intenção do repositório é coletar padrões que comumente ocorrem na especificação de sistemas concorrentes e reativos.

Os dados do repositório foram coletados para uma pesquisa realizada em 1998. O contexto dessa pesquisa era sobre como as pessoas estavam utilizando especificações formais. Foram solicitados exemplos de requisitos que pudessem ser verificados com ferramentas de verificação de estado finito. Além disso, foram coletadas uma série de especificações presentes na literatura. Desde então, são feitas atualizações periódicas para compor essa base de dados (Dwyer; Avrunin; Corbett, 1999).

A partir dos dados retirados do repositório “Padrões de Especificação Temporal”, foi criado um conjunto de contendo propriedades temporais CTL. Originalmente os requisitos que geram as propriedades do repositório estão escritos na língua inglesa. Como o escopo desse trabalho utiliza a língua portuguesa do Brasil como linguagem natural para compor as sentenças, os requisitos originais foram traduzidas para o português. Além disso, as propriedades CTL, foram reescritas manualmente em UPPAAL, visto que essa é a linguagem alvo da tradução.

Para que esse conjunto fosse criado, foi preciso realizar um pré processamento manual. No pré processamento foram retiradas as propriedades que não possuíam um requisito associado. Além disso, as sentenças foram escritas de forma que tanto o tradutor neural quanto o tradutor baseado em regras pudessem compreender.

As Tabelas 4.7 e 4.8 apresentam uma amostragem do conjunto de dados contendo as propriedades e os requisitos escritos em português.

Tabela 4.7 – Propriedades e Requisitos - Dwyer 1

Requisito	Propriedade
Sempre que o título estiver na Lista , implica que inevitavelmente o livro será encontrado	A[] (titulo.Lista imply A<> (livro.encontrado))
Sempre que o autor estiver na Lista , implica que inevitavelmente o livro será encontrado	A[] (autor.Lista imply A<> (livro.encontrado))
Sempre que o assunto estiver na Lista , implica que inevitavelmente o livro será encontrado	A[] (assunto.Lista imply A<> (livro.encontrado))
Sempre que o número ISBN estiver na lista , implica que inevitavelmente o livro será encontrado	A[] (numero_ISBN.Lista imply A<> (livro.encontrado))
Sempre que o livro não for cobrado , implica que inevitavelmente o livro não será finalizado	A[] (! (livro.cobrado) imply A<> (! (livro.finalizado)))
Sempre que nameinList3 for igual a false , implica que inevitavelmente a senha estará sempre correta	A[] (nameinList3 ==false imply A<> (senha.correta)
Se o estado do livro é igual a espera posteriormente o estado do livro não será igual a cobrado	estado_livro ==espera –> ! (estado_livro ==cobrado) !

Fonte: Elaborado pelo Autor.

Tabela 4.8 – Propriedades e Requisitos - Dwyer 2

Requisito	Propriedade
Se o titlo não estiver na lista, posteriormente o livro não poderá ser encontrado	! (titulo.Lista) –> ! (livro.encontrado)
Se o assunto não estiver na lista, posteriormente o livro não poderá ser encontrado	! (assunto.Lista) –> ! (livro.encontrado)
Se o número não estiver na lista, posteriormente o livro não poderá ser encontrado	! (numero.Lista) –> ! (livro.encontrado)
Se o usuário inserir dados corretos posteriormente o botão OK é habilitado	usuario.inserir_dados_corretos –> botao_OK.habilitado
Se o login não é validado, posteriormente o aplicativo não deverá ser inicializado e o botão OK não deverá ser habilitado	! (login.validado) –> ! (aplicativo.inicializado) and ! (botao_OK.habilitado)
Se o login é gerente de projeto ou login é administrador de banco de dados posteriormente os botões não deverão ser habilitados	login.gerente_projeto or login.administrador_banco_dados –> ! (botoes.habilitados)
Caso a informação seja inserida no campo de entrada e o botão Pesquisar seja pressionado, posteriormente todas as informações serão recuperadas	informacao.inserida_campo_entrada and botao_Pesquisar.pressionado –> todas_informacoes.recuperadas
Quando a informação é inserida no campo de entrada, posteriormente o comando de atualização está habilitado	informacao . inserida_campo_entrada imply comando_atualizacao . habilitado
Se o login for igual a Administração de projetos, posteriormente o menu Relatórios Genéricos deve estar disponível	login ==Administracao_projetos –> menu_Relatorios_Genericos.disponivel
Se o MS estiver em chamada então inevitavelmente o msmeasstate deverá ser meason	MS.chamada imply A<> (msmeasstate.meason)
Se BsHandoffAccept for recebido, implica que inevitavelmente o MS transmissor será desligado	BsHandoffAccept.recebido imply A<> (MS.transmissor_desligado)

Fonte: Elaborado pelo Autor.

4.4 CONJUNTO DE DADOS

Para que fosse possível realizar os experimentos de tradução, foi construído um conjunto de dados contendo os requisitos escritos em português e as propriedades em UPPAAL que foram apresentadas nos estudos de caso das seções 4.1 , 4.2, 4.3.

A partir de uma análise visual foi identificado que muitos pares de requisitos e propriedades definidos na coleção de Dwyer estavam descontextualizadas, ou seja, não foi possível compreender visualmente o sentido das propriedades visto que muitas variáveis utilizadas nas propriedades não eram mencionadas nos requisitos correspondentes. Para que o entendimento dessas propriedades fosse completo seria necessário

analisar as informações descritas a partir de todo o contexto do sistema ao qual os requisitos e as propriedades pertencem.

Um exemplo de caso de propriedade e requisito fora de contexto é o requisito em inglês: "*Contingency Guidance shall command an ET separation.*", traduzido para o português: "*A Orientação de Contingência deve comandar uma separação ET.*" A correspondência desse requisito em CTL foi definido da seguinte maneira na base de Dwyer: $AG(cs.cont\_3EO\_start \,\&\, cg.finished \,\&\, (cg.r = reg1 \rightarrow cond\_29) \,\&\, (cg.r = reg2 \rightarrow cond\_24 \& cond\_26) \,\&\, (cg.r = reg3 \rightarrow cg.alpha\_ok \,\&\, (ABS\_alf\_err\_LT\_alf\_sep\_err \rightarrow cond\_20b)) \,\&\, (cg.r = reg4 \rightarrow cond\_18 \& q\_orb\_LT\_0) \,\&\, (cg.r = reg102 \rightarrow pre\_sep) \rightarrow et\_sep\_cmd|cg.et\_sep\_man\_initiate|cg.early\_sep|cg.emerg\_sep))$

No exemplo apresentado fica evidente que variáveis utilizadas na propriedade como *cond_18*,*reg3*, entre outras, não foram definidas na construção da sentença do requisito. Essa característica dificulta a tradução visto que não existe um correspondente associado as variáveis utilizadas na propriedade. Para minimizar os ruídos, esses pares de requisitos e propriedades foram eliminados do conjunto de dados.

Na abordagem de tradução baseada em regras os requisitos foram agrupados e testados por caso de estudo, visto que esse algoritmo utiliza os modelos (se houver) contendo os diagramas de Blocos, Estado e Requisitos para auxiliar no processo de tradução. Já a abordagem de tradução baseada em redes neurais utilizou o conjunto de dados levantados contendo os pares de requisitos e propriedades de todos os casos de estudo apresentados. A Tabela 4.9 apresenta um resumo da composição do conjunto de dados.

Tabela 4.9 – Conjunto de Dados

Abordagem	Requisitos	Quantidade de requisitos
Tradutor baseado em regras	Jantar dos Filósofos	16
Tradutor baseado em regras	Protocolo Fischer	9
Tradutor baseado em regras	Base de Dwyer	166
Tradutor baseado em redes neurais - LSTM	Jantar + Fischer + Dwyer	191
Tradutor baseado em redes neurais - Transformer	Jantar + Fischer + Dwyer	191

Fonte: Elaborado pelo Autor.

No próximo capítulo serão apresentados os resultados obtidos com os métodos aplicados bem como a avaliação desses resultados.

4.5 RESULTADOS E AVALIAÇÃO

Para avaliar os resultados obtidos foi utilizada a métrica de precisão (*accuracy*). A métrica de precisão é definida por Mikhail e Ackermann (1976) como sendo o grau de proximidade de uma estimativa com valor verdadeiro. No caso da tradução foram comparadas as propriedades resultantes do processo de tradução e as propriedades que deveriam ser obtidas pelo processo.

Para este trabalho foram definidas três classificações de tradução:

- BOA: Tradução com precisão maior ou igual 0,5 e avaliação manual que garante que o resultado da tradução da propriedade tenha sido igual ou muito próximo da propriedade original.
- REGULAR: Tradução com qualquer valor de precisão e avaliação manual que garante que o resultado da tradução da propriedade tenha alguma semelhança com a propriedade original.
- RUIM: Tradução com precisão menor que 0,5 e avaliação manual constando que as traduções ficaram totalmente divergente da propriedade original.

Nas próximas seções serão apresentados os resultados obtidos através tradutor baseado em redes neurais e os resultados obtidos pelo tradutor baseado em regras, será adicionalmente apresentado um breve comparativo sobre essas duas abordagens utilizadas.

4.5.1 Tradutor baseado em redes neurais - Modelo decodificador codificador LSTM

Nos experimentos utilizando a rede codificador decodificador do tipo LSTM, sem mecanismo de atenção, foram executados 3 conjuntos de testes alternando a quantidade de neurônios da camada LSTM do codificador e do decodificador, além da quantidade de épocas.

O primeiro conjunto de treino foi realizado com 2000 épocas, 64 neurônios para a camada LSTM do decodificador e 64 neurônios para a camada LSTM do codificador. O segundo conjunto de treino foi realizado com 3000 épocas e utilizou 64 neurônios para a camada LSTM do decodificador e 64 neurônios para a camada LSTM do codificador. Já para o terceiro conjunto de treino, a quantidade de épocas foi igual a 3000 sendo utilizados 256 neurônios para a camada LSTM do decodificador e 256 neurônios para a camada LSTM do codificador.

A validação foi feita utilizando o método de validação cruzada, onde o conjunto de dados é dividido em *n* partes, sendo que uma dessas partes é utilizada para teste e as demais são utilizadas para o treino. Esse ciclo se repete até todas as partes serem utilizadas para teste.

Tabela 4.10 – Resultados - Modelo decodificador codificador LSTM

Base Teste	Base Treino	Qt. Base Treino	Qt. Base Teste	Épocas	Neurônios Dec. e Encod.	Acc < 0,5	Acc
parte 1	partes 2 e 3	127 registros	64 registros	2000	64	todos os registros	0 <= Acc < 0,15
parte 2	partes 1 e 3	127 registros	64 registros	2000	64	todos os registros	0 <= Acc < 0,05
parte 3	partes 1 e 2	128 registros	63 registros	2000	64	todos registros	0 <= Acc < 0,10

Fonte: Elaborado pelo Autor.

A decisão de realizar três conjuntos de testes foi devido à baixa qualidade nos resultados obtidos por essa arquitetura de rede neural para a tradução de propriedades. Não foram obtidos resultados satisfatórios, pois todas as traduções apresentaram uma precisão menor do que 0,5. Além disso, os resultados foram avaliados manualmente e todas as traduções estavam totalmente divergentes da propriedade original.

A Tabela 4.10 apresenta os resultados obtidos no melhor conjunto de teste entre os três conjuntos realizados. Nesse conjunto de teste foram utilizados 64 neurônios para a camada LSTM do decodificador e 64 neurônios para a camada LSTM do codificador e a rede foi treinada com 2000 épocas.

Pela Tabela 4.10 é possível observar que mesmo para o melhor resultado obtido, a precisão das traduções é inferior a 0,5. A Tabela 4.11 apresenta alguns exemplos das traduções onde é possível observar o resultado obtido e o resultado esperado da tradução, além da classificação desses resultados.

Tabela 4.11 – Tradução - Modelo decodificador codificador LSTM

Requisito	Tradução	Resultado esperado	Classificação
sempre que currphase = normalrun, implica que inevitavelmente currphase = criticalrun	–>(.!!!–>((	a[] (currphase == normalrun imply a<> (currphase == criticalrun))	RUIM
se os dados de origem não são carregados, posteriormente a janela de graficos poderá ser aberta	====menu_relatorios_genericos-menu_relatorios_genericosmenu_relatorios_genericosdisponiveldisponiveldisponivel–>–>	! (dados_origem . carregados) –> janela_graficos . aberta	RUIM

Fonte: Elaborado pelo Autor.

Através da análise manual e dos resultados da precisão obtidos, todos as traduções para essa rede neural foram consideradas inapropriadas e por tando foram classificadas como “RUIM”.

4.5.2 Tradutor baseado em redes neurais - Transformer

A tradução com a rede *transformer*, foi realizada apenas para um conjunto de teste. Nesse conjunto de teste foram ajustados os seguintes parâmetros:

- d_model = 128;
- nb_layers = 4 ;
- ffn_units = 512;
- épocas = 2000.

A validação foi feita utilizando o método de validação cruzada, onde o conjunto de dados é dividido em *n* partes, sendo que uma dessas partes é utilizada para teste e as demais partes são utilizadas para o treino. Esse ciclo se repete até todas as partes serem utilizadas para teste.

A primeira parte da base separada como base de teste, contendo 64 registros teve como resultado apenas 17 registros tiveram precisão menor que 0,5 A segunda parte da base separada para teste, também contém 64 registros. Dos 64 registros apenas 19 registros tiveram precisão menor que 0,5. A terceira parte da base separada para teste, contém 63 registros. Dos 63 registros apenas 17 registros tiveram precisão menor que 0,5.

Os resultados são apresentados na Tabela 4.12 assim como a classificação dos resultados.

Tabela 4.12 – Resultados - Transformer

Base Teste	Base Treino	Qt. Base Treino	Qt. Base Teste	Épocas	Acc >= 0,5	Acc < 0,5	BOM	REG.	RUIM
parte 1	partes 2 e 3	127 registros	64 registros	2000	31 registros	17 registros	19 registros	31 registros	14 registros
parte 2	partes 1 e 3	127 registros	64 registros	2000	23 registros	19 registros	15 registros	38 registros	11 registros
parte 3	partes 1 e 2	128 registros	63 registros	2000	28 registros	17 registros	21 registros	32 registros	10 registros

Fonte: Elaborado pelo Autor.

Visto que o treino com a rede neural *transformer* apresentou resultados com classificação diferentes de RUIM, foi possível realizar uma análise mais detalhada dos resultados dessa abordagem para os casos de estudo. A seguir serão apresentados os resultados para os casos de estudo: Jantar dos filósofos, protocolo Fischer, e uma

amostragem da base de Dwyer. As Tabelas 4.13 e 4.14 apresentam os resultados para o caso de estudo: Jantar dos Filósofos.

Tabela 4.13 – Tradução baseada em redes neurais - Jantar dos Filósofos 1

Requisito	Propriedade	Resultado Tradução	Classificação
Nunca ocorrerá deadlock	$A[]\ not\ deadlock$	A[] not (botao_Atualizar . habilitado)	RUIM
o filosofo 1 e o filosofo 2 nunca comem ao mesmo tempo	A[] not (Filosofo1. comer and Filosofo2.comer)	A[] not (Filosofo2 . comem and Filosofo3 . comem)	REGULAR
o filosofo 2 e o filosofo 3 nunca comem ao mesmo tempo	A[] not (Filosofo2. comer and Filosofo3.comer)	A[] not (Filosofo3 . comem and Filosofo4 . comem)	REGULAR
o filosofo 3 e o filosofo 4 nunca comem ao mesmo tempo	A[] not (Filosofo3.comer and Filosofo4.comer)	A[] not (Filosofo4 . comem and Filosofo5 . comem)	REGULAR
o filosofo 4 e o filosofo 5 nunca comem ao mesmo tempo	A[] not (Filosofo4. comer and Filosofo5.comer)	A[] not (Filosofo5 . comem and Filosofo1 . comem)	REGULAR
o filosofo 5 e o filosofo 1 nunca comem ao mesmo tempo	A[] not (Filosofo5. comer and Filosofo1.comer)	A[] not (filosofo5 . comendo and filosofo1 . comendo)	BOM
sempre que o filosofo 5 estiver comendo implica que o filosofo 4 e o filosofo 1 não estejam comendo	A[] Filosofo5.comer imply ! Filosofo4.comer and ! Filosofo1.comer	A[] (Filosofo5 . comendo imply ! (Filosofo4 . irao_comer and Filosofo1 . irao_comer))	BOM
sempre que o filosofo 1 estiver comendo implica que o filosofo 5 e o filosofo 2 não estejam comendo	A[] Filosofo1.comer imply ! Filosofo5.comer and ! Filosofo2.comer	A[] (Filosofo1 . comem and Filosofo2 . comem)	REGULAR
sempre que o filosofo 4 estiver comendo implica que o filosofo 3 e o filosofo 5 não estejam comendo	A[] Filosofo4.comer imply ! Filosofo3.comer and ! Filosofo5.comer	A[] (Filosofo5 . comendo imply ! (Filosofo4 . irao_comer and Filosofo1 . irao_comer))	REGULAR
sempre que o filosofo 2 estiver comendo implica que o filosofo 3 e o filosofo 1 não estejam comendo	A[] Filosofo2.comer imply ! Filosofo3.comer and ! Filosofo1.comer	A[] (Filosofo1 . comendo imply ! (Filosofo5 . irao_comer and Filosofo2 . irao_comer))	REGULAR
Se o filósofo 5 estiver com fome, futuramente o filósofo 5 irá comer	Filosofo5.fome $\rightarrow$ Filosofo5.comer	Filosofo5 . fome $\rightarrow$ Filosofo5 . comer	BOM

Fonte: Elaborado pelo Autor.

Tabela 4.14 – Tradução baseada em redes neurais - Jantar dos Filósofos 2

sempre que o filosofo 3 estiver comendo implica que o filosofo 4 e o filosofo 2 não estejam comendo	A[] Filosofo3.comer imply ! Filosofo4.comer and ! Filosofo2.comer	A[] (Filosofo2 . comendo imply ! (Filosofo3 . irao_comer and Filosofo1 . irao_comer))	REGULAR
Se o filósofo 1 estiver com fome, futuramente o filósofo 1 irá comer	Filosofo1.fome $\rightarrow$ Filosofo1.comer	Filosofo5 . fome $\rightarrow$ Filosofo5 . comer	REGULAR
Se o filósofo 2 estiver com fome, futuramente o filósofo 2 irá comer	Filosofo2.fome $\rightarrow$ Filosofo2.comer	Filosofo1 . fome $\rightarrow$ Filosofo1 . comer	REGULAR
Se o filósofo 3 estiver com fome, futuramente o filósofo 3 irá comer	Filosofo3.fome $\rightarrow$ Filosofo3.comer	Filosofo4 . fome $\rightarrow$ Filosofo4 . comer	REGULAR
Se o filósofo 4 estiver com fome, futuramente o filósofo 4 irá comer	Filosofo4.fome $\rightarrow$ Filosofo4.comer	Filosofo3 . fome $\rightarrow$ Filosofo3 . comer	REGULAR

Fonte: Elaborado pelo Autor.

Analisando as Tabelas 4.13 e 4.14 do caso de estudo: Jantar dos filósofos, apenas três resultados de tradução se mostraram compatíveis com a propriedade original, um foi classificado como ruim e os demais registros obtiveram uma tradução aproximada, portanto foram classificados com regular. As próximas Tabelas 4.15 e 4.16 apresentam o resultado para caso de estudo: protocolo de Fischer.

Tabela 4.15 – Tradução baseada em redes neurais - Protocolo de Fischer 1

Requisito	Propriedade	Resultado Tradução	Classificação
Nunca ocorrerá deadlock	$A[]\ not\ deadlock$	A[] not (botao_Atualizar . habilitado)	RUIM
Sempre que o processo 1 requisitar entrar na seção critica, obriga que o processo 1 inevitavelmente entre no local de espera	processo1.req –> processo1.espera	processo1.req –> processo1.regiao_critica	REGULAR

Fonte: Elaborado pelo Autor.

Tabela 4.16 – Tradução baseada em redes neurais - Protocolo de Fischer 2

Requisito	Propriedade	Resultado Tradução	Classificação
Sempre que o processo 1 requisitar entrar na seção critica, obriga que o processo 1 inevitavelmente entre na seção crítica	processo1.req –> processo1.regiao_critica	processo3.req –> processo3.regiao_critica	REGULAR
Sempre que o processo 2 requisitar entrar na seção crítica, obriga que o processo 2 inevitavelmente entre no local de espera	processo2.req –> processo2.espera	processo3.req –> processo3.regiao_critica	REGULAR
Sempre que o processo 3 requisitar entrar na seção crítica, obriga que o processo 3 inevitavelmente entre no local de espera	processo3.req –> processo3.espera	processo1.req –> processo1.espera	REGULAR
Sempre que o processo 4 requisitar entrar na seção crítica, obriga que o processo 4 inevitavelmente entre no local de espera	processo4.req –> processo4.espera	processo1.req –> processo1.espera	REGULAR
Sempre que o processo 4 requisitar entrar na seção crítica, obriga que o processo 4 inevitavelmente entre na seção crítica	processo4.req –> processo4.regiao_critica	processo1.req –> processo1.espera	REGULAR
Sempre que o processo 3 requisitar entrar na seção crítica, obriga que o processo 3 inevitavelmente entre na seção crítica	processo3.req –> processo3.regiao_critica	processo1.req –> processo1.espera	REGULAR
Sempre que o processo 2 requisitar entrar na seção crítica, obriga que o processo 2 inevitavelmente entre na seção crítica	processo2.req –> processo2.regiao_critica	processo1.req –> processo1.espera	REGULAR

Fonte: Elaborado pelo Autor.

Uma amostragem dos resultados e a classificação para a base de Dwyer tam-

bém são exibidas nas Tabelas 4.17 e 4.18 .

Tabela 4.17 – Tradução baseada em redes neurais - Dwyer 1

Requisito	Propriedade	Resultado Tradução	Classificação
Sempre que o título estiver na Lista, implica que inevitavelmente o livro será encontrado	A[] (titulo.Lista imply A<> (livro.encontrado))	A[] (titulo . Lista imply A<> (livro . encontrado))	BOM
Sempre que o autor estiver na Lista, implica que inevitavelmente o livro será encontrado	A[] (autor.Lista imply A<> (livro.encontrado))	A[] (autor . Lista imply A<> (livro . encontrado))	BOM
Sempre que o assunto estiver na Lista , implica que inevitavelmente o livro será encontrado	A[] (assunto.Lista imply A<> (livro.encontrado))	A[] (assunto . Lista imply A<> (livro . encontrado))	BOM
Sempre que o número ISBN estiver na lista , implica que inevitavelmente o livro será encontrado	A[] (numero_ISBN.Lista imply A<> (livro.encontrado))	A[] (numero_ISBN . Lista imply A<> (livro . encontrado))	BOM
Sempre que o livro não for cobrado , implica que inevitavelmente o livro não será finalizado	A[] (! (livro.cobrado) imply A<> (! (livro.finalizado)))	A[] (! (livro . cobrado) imply A<> (! (livro . finalizado)))	BOM
Sempre que nameinList3 for igual a false , implica que inevitavelmente a senha estará sempre correta	A[] (nameinList3 ==false imply A<> (senha.correta)	A[] (currPhase == . NormalRun imply A<> (currPhase == . CriticalRun))	RUIM
Se o estado do livro é igual a espera posteriormente o estado do livro não será igual a cobrado	estado_livro ==espera –> ! (estado_livro ==cobrado)	estado_livro ==espera –> ! (estado_livro ==cobrado)	BOM
Se o titlo não estiver na lista, posteriormente o livro não poderá ser encontrado	! (titulo.Lista) –> ! (livro.encontrado)	! (assunto . Lista) imply ! (livro . encontrado)	REGULAR
Se o assunto não estiver na lista, posteriormente o livro não poderá ser encontrado	! (assunto.Lista) –> ! (livro.encontrado)	! (numero . Lista) –> ! (livro . encontrado)	REGULAR

Fonte: Elaborado pelo Autor.

Tabela 4.18 – Tradução baseada em redes neurais - Dwyer 2

Requisito	Propriedade	Resultado Tradução	Classificação
Se o número não estiver na lista, posteriormente o livro não poderá ser encontrado	! (numero.Lista) –> ! (livro.encontrado)	! (autor . Lista) –> ! (livro . encontrado)	REGULAR
Se o usuário inserir dados corretos posteriormente o botão OK é habilitado	usuario.inserir_dados_corretos –> botao_OK.habilitado	informacao . inserida_campo_entrada –> comando_atualizacao . habilitado	RUIM
Se o login não é validado, posteriormente o aplicativo não deverá ser inicializado e o botão OK não deverá ser habilitado	! (login.validado) –> ! (aplicativo.inicializado) and ! (botao_OK.habilitado)	! (! (! (! (! (! (numero . Lista) –> ! (livro . encontrado)	RUIM
Se o login é gerente de projeto ou login é administrador de banco de dados posteriormente os botões não deverão ser habilitados	login.gerente_projeto or login.administrador_banco_dados –> ! (botoes.habilitados)	login . gerente_projeto or login . administrador_banco_dados imply ! (botoes . habilitados)	BOM
Caso a informação seja inserida no campo de entrada e o botão Pesquisar seja pressionado, posteriormente todas as informações serão recuperadas	informacao.inserida_campo_entrada and botao_Pesquisar.pressionado –> todas_informacoes.recuperadas	operacao_desbloqueio . solicitada –> portas . desbloqueadas	RUIM
Quando a informação é inserida no campo de entrada, posteriormente o comando de atualização está habilitado	informacao.inserida-informacao . inserida_campo_entrada imply comando_atualizacao . habilitado	informacao . inserida_campo_entrada imply comando_atualizacao . habilitado	BOM
Se o login for igual a Administração de projetos, posteriormente o menu Relatórios Genéricos deve estar disponível	login ==Administracao_projetos –> menu_Relatorios_Genericos.disponivel	login . gerente_projeto or login . administrador_banco_dados imply ! (botoes . habilitados)	REGULAR
Se o MS estiver em chamada então inevitavelmente o msmeasstate deverá ser meason	MS.chamada imply A<> (msmeasstate.meason)	MS . chamada imply A<> (msmeasstate . meason)	BOM
Se BsHandoffAccept for recebido, implica que inevitavelmente o MS transmissor será desligado	BsHandoffAccept. recebido imply A<> (MS.transmissor_desligado)	BsHandoffAccept . recebido imply A<> (MS . transmissor_desligado)	BOM

Fonte: Elaborado pelo Autor.

4.5.3 Tradutor baseado em regras

O Algoritmo de tradução ou tradutor baseado em regras, foi aplicado nos 3 casos de estudo: Jatar do Filósofos, Protocolo de Fischer e Base de Dwyer. Os resultados e a classificação dos resultados para o caso de estudo: Jantar dos Filósofos podem ser observados nas Tabelas 4.19 e 4.20

Tabela 4.19 – Tradução baseada em regras - Jantar dos Filósofos 1

Requisito	Propriedade	Resultado Tradução	Classificação
Nunca ocorrerá deadlock	$A[]\ not\ deadlock$	$A[]\ not\ deadlock$	BOM
o filosofo 1 e o filosofo 2 nunca comem ao mesmo tempo	A[] not (Filosofo1.comer and Filosofo2.comer)	A[] not (Filosofo1.comer and Filosofo2.comer)	BOM
o filosofo 2 e o filosofo 3 nunca comem ao mesmo tempo	A[] not (Filosofo2.comer and Filosofo3.comer)	A[] not (Filosofo2.comer and Filosofo3.comer)	BOM
o filosofo 3 e o filosofo 4 nunca comem ao mesmo tempo	A[] not (Filosofo3.comer and Filosofo4.comer)	[] not (Filosofo3.comer and Filosofo4.comer)	BOM
o filosofo 4 e o filosofo 5 nunca comem ao mesmo tempo	A[] not (Filosofo4.comer and Filosofo5.comer)	A[] not (Filosofo4.comer and Filosofo5.comer)	BOM
o filosofo 5 e o filosofo 1 nunca comem ao mesmo tempo	A[] not (Filosofo5.comer and Filosofo1.comer)	A[] not (Filosofo5.comer and Filosofo1.comer)	BOM
sempre que o filosofo 5 estiver comendo implica que o filosofo 4 e o filosofo 1 não estejam comendo	A[] Filosofo5.comer imply ! Filosofo4.comer and ! Filosofo1.comer	A[] Filosofo5.comer imply ! Filosofo4.comer and ! Filosofo1.comer	BOM
sempre que o filosofo 1 estiver comendo implica que o filosofo 5 e o filosofo 2 não estejam comendo	A[] Filosofo1.comer imply ! Filosofo5.comer and ! Filosofo2.comer	A[] Filosofo1.comer imply ! Filosofo5.comer and ! Filosofo2.comer	BOM
sempre que o filosofo 4 estiver comendo implica que o filosofo 3 e o filosofo 5 não estejam comendo	A[] Filosofo4.comer imply ! Filosofo3.comer and ! Filosofo5.comer	A[] Filosofo4.comer imply ! Filosofo3.comer and ! Filosofo5.comer	BOM
sempre que o filosofo 2 estiver comendo implica que o filosofo 3 e o filosofo 1 não estejam comendo	A[] Filosofo2.comer imply ! Filosofo3.comer and ! Filosofo1.comer	A[] Filosofo2.comer imply ! Filosofo3.comer and ! Filosofo1.comer	BOM

Fonte: Elaborado pelo Autor.

Tabela 4.20 – Tradução baseada em regras - Jantar dos Filósofos 2

sempre que o filosofo 3 estiver comendo implica que o filosofo 4 e o filosofo 2 não estejam comendo	A[] Filosofo3.comer imply ! Filosofo4.comer and ! Filosofo2.comer	A[] Filosofo3.comer imply ! Filosofo4.comer and ! Filosofo2.comer	BOM
Se o filósofo 1 estiver com fome, futuramente o filósofo 1 irá comer	Filosofo1.fome $\rightarrow$ Filosofo1.comer	Filosofo1.fome $\rightarrow$ Filosofo1.comer	BOM
Se o filósofo 2 estiver com fome, futuramente o filósofo 2 irá comer	Filosofo2.fome $\rightarrow$ Filosofo2.comer	Filosofo2.fome $\rightarrow$ Filosofo2.comer	BOM
Se o filósofo 3 estiver com fome, futuramente o filósofo 3 irá comer	Filosofo3.fome $\rightarrow$ Filosofo3.comer	Filosofo3.fome $\rightarrow$ Filosofo3.comer	BOM
Se o filósofo 4 estiver com fome, futuramente o filósofo 4 irá comer	Filosofo4.fome $\rightarrow$ Filosofo4.comer	Filosofo4.fome $\rightarrow$ Filosofo4.comer	BOM
Se o filósofo 5 estiver com fome, futuramente o filósofo 5 irá comer	Filosofo5.fome $\rightarrow$ Filosofo5.comer	Filosofo5.fome $\rightarrow$ Filosofo5.comer	BOM

Fonte: Elaborado pelo Autor.

Já os resultados do caso de estudo: Protocolo de Fischer podem ser observados nas Tabelas 4.21 e 4.22.

Tabela 4.21 – Tradução baseada em regras - Protocolo de Fischer 1

Requisito	Propriedade	Resultado Tradução	Classificação
Sempre que o processo 1 requisitar entrar na seção critica, obriga que o processo 1 inevitavelmente entre no local de espera	processo1.req –> processo1.espera	processo1.req –> processo1.espera	BOM
Nunca ocorrerá deadlock	$A[]\ not\ deadlock$	$A[]\ not\ deadlock$	BOM

Fonte: Elaborado pelo Autor.

Tabela 4.22 – Tradução baseada em regras - Protocolo de Fischer 2

Requisito	Propriedade	Resultado Tradução	Classificação
Sempre que o processo 1 requisitar entrar na seção critica, obriga que o processo 1 inevitavelmente entre na seção crítica	processo1.req –> processo1.regiao_critica	processo1.req –> processo1.regiao_critica	BOM
Sempre que o processo 4 requisitar entrar na seção crítica, obriga que o processo 4 inevitavelmente entre no local de espera	processo4.req –> processo4.espera	processo4.req –> processo4.espera	BOM
Sempre que o processo 4 requisitar entrar na seção crítica, obriga que o processo 4 inevitavelmente entre na seção crítica	processo4.req –> processo4.regiao_critica	processo4.req –> processo4.regiao_critica	BOM
Sempre que o processo 3 requisitar entrar na seção crítica, obriga que o processo 3 inevitavelmente entre na seção crítica	processo3.req –> processo3.regiao_critica	processo3.req –> processo3.regiao_critica	BOM
Sempre que o processo 2 requisitar entrar na seção crítica, obriga que o processo 2 inevitavelmente entre na seção crítica	processo2.req –> processo2.regiao_critica	processo2.req –> processo2.regiao_critica	BOM
Sempre que o processo 2 requisitar entrar na seção crítica, obriga que o processo 2 inevitavelmente entre no local de espera	processo2.req –> processo2.espera	processo2.req –> processo2.espera	BOM
Sempre que o processo 3 requisitar entrar na seção crítica, obriga que o processo 3 inevitavelmente entre no local de espera	processo3.req –> processo3.espera	processo3.req –> processo3.espera	BOM

Fonte: Elaborado pelo Autor.

Para o terceiro caso de estudo: Base de Dwyer, a amostragem dos resultados

são apresentados nas Tabelas 4.23 e 4.24.

Tabela 4.23 – Tradução baseada em regras - Dwyer 1

Requisito	Propriedade	Resultado Tradução	Classificação
Sempre que o título estiver na Lista , implica que inevitavelmente o livro será encontrado	A[] (titulo.Lista imply A<> (livro.encontrado))	A[] (titulo.Lista imply A<> (livro.encontrado))	BOM
Sempre que o autor estiver na Lista , implica que inevitavelmente o livro será encontrado	A[] (autor.Lista imply A<> (livro.encontrado))	A[] (autor.Lista imply A<> (livro.encontrado))	BOM
Sempre que o assunto estiver na Lista , implica que inevitavelmente o livro será encontrado	A[] (assunto.Lista imply A<> (livro.encontrado))	A[] (assunto.Lista imply A<> (livro.encontrado))	BOM
Sempre que o número ISBN estiver na lista , implica que inevitavelmente o livro será encontrado	A[] (numero_ISBN.Lista imply A<> (livro.encontrado))	A[] (numero_ISBN.Lista imply A<> (livro.encontrado))	BOM
Sempre que o livro não for cobrado , implica que inevitavelmente o livro não será finalizado	A[] (! (livro.cobrado) imply A<> (! (livro.finalizado)))	A[] (! (livro.cobrado) imply A<> (! (livro.finalizado)))	BOM
Sempre que nameinList3 for igual a false , implica que inevitavelmente a senha estará sempre correta	A[] (nameinList3 ==false imply A<> (senha.correta)	A[] (nameinList3==false imply A<> (senha.correta)	BOM
Se o estado do livro é igual a espera posteriormente o estado do livro não será igual a cobrado	estado_livro ==espera –> ! (estado_livro ==cobrado)	estado_livro==espera –> ! (estado_livro == cobrado)	BOM
Se o titlo não estiver na lista, posteriormente o livro não poderá ser encontrado	! (titulo.Lista) –> ! (livro.encontrado)	! (titulo.Lista) –> ! (livro.encontrado)	BOM
Se o assunto não estiver na lista, posteriormente o livro não poderá ser encontrado	! (assunto.Lista) –> ! (livro.encontrado)	! (assunto.Lista) –> ! (livro.encontrado))	BOM

Fonte: Elaborado pelo Autor.

Tabela 4.24 – Tradução baseada em regras - Dwyer 2

Requisito	Propriedade	Resultado Tradução	Classificação
Se o número não estiver na lista, posteriormente o livro não poderá ser encontrado	! (numero.Lista) –> ! (livro.encontrado)	! (numero.Lista) –> ! (livro.encontrado)	BOM
Se o usuário inserir dados corretos posteriormente o botão OK é habilitado	usuario.inserir_dados_corretos –> botao_OK.habilitado	usuario.inserir_dados_corretos –> botao_OK.habilitado	BOM
Se o login não é validado, posteriormente o aplicativo não deverá ser inicializado e o botão OK não deverá ser habilitado	! (login.validado) –> ! (aplicativo.inicializado) and ! (botao_OK.habilitado)	! (login.validado) –> ! (aplicativo.inicializado) and ! (botao_OK.habilitado)	BOM
Se o login é gerente de projeto ou login é administrador de banco de dados posteriormente os botões não deverão ser habilitados	login.gerente_projeto or login.administrador_banco_dados –> ! (botoes.habilitados)	login.administrador_banco gerente projeto or login.administrador_banco dados –> ! (botoes.habilitados)	REGULAR
Caso a informação seja inserida no campo de entrada e o botão Pesquisar seja pressionado, posteriormente todas as informações serão recuperadas	informacao.inserida_campo_entrada and botao_Pesquisar.pressionado –> todas_informacoes.recuperadas	informacao- .inserida_campo_entrada and botao_Pesquisar.pressionado –> todas_informacoes.recuperadas	BOM
Quando a informação é inserida no campo de entrada, posteriormente o comando de atualização está habilitado	informacao.inserida-informacao . inserida_campo_entrada imply comando_atualizacao . habilitado	informacao.inserida_campo_entrada imply comando_atualizacao.habilitado	BOM
Se o login for igual a Administração de projetos, posteriormente o menu Relatórios Genéricos deve estar disponível	login ==Administracao_projetos –> menu_Relatorios_Genericos.disponivel	login == Administracao_projetos –> menu_Relatorios_Genericos.disponivel	BOM
Se o MS estiver em chamada então inevitavelmente o msmeasstate deverá ser meason	MS.chamada imply A<> (msmeasstate.meason)	MS . chamada imply A<> (msmeasstate . meason)	BOM
Se BsHandoffAccept for recebido, implica que inevitavelmente o MS transmissor será desligado	BsHandoffAccept. recebido imply A<> (MS.transmissor_desligado)	BsHandoffAccept . recebido imply A<> (MS . transmissor_desligado)	BOM

Fonte: Elaborado pelo Autor.

Para base de Dwyer, dos 166 requisitos traduzidos, 42 registros foram classifi-

cados como BOM, 15 registros foram classificados como RUIM, e 109 registros foram considerados com regular.

4.5.4 Considerações

Apesar da rede LSTM não ter se mostrado eficiente na tradução, a rede *transformer* obteve um resultado satisfatório considerando que o conjunto de dados utilizado foi limitado em relação a qualidade por possuir pouca quantidades de sentenças e palavras.

Muitos fatores podem interferir na qualidade da tradução como: quantidade de épocas, tamanho do conjunto de dados, variedade de palavras e estruturas do conjunto de dados, além disso outros fatores como a tradução de requisitos do inglês para o português realizada manualmente permite que informações possam ser perdidas e consequentemente deixe os dados defasados em relação a sua qualidade.

Apesar disso, diante dos resultados obtidos, foi possível verificar que a tradução baseada em redes neurais pode ser utilizada para tradução de um requisito escrito em Português para uma propriedade temporal formal na linguagem UPPAAL. Uma das vantagens em utilizar a tradução baseada em redes neurais é que não é necessário definir regras sintáticas para a construção das sentenças, já no tradutor baseado em regras, a estrutura deve ser definida pelo programador.

Para os casos de estudo apresentados, o tradutor baseado em redes neurais, conseguiu traduzir as sentenças, no entanto, houveram algumas trocas de palavras na composição das propriedades, por exemplo: “ *o filosofo 4 e o filosofo 5 nunca comem ao mesmo tempo* ” onde a propriedade original é: A[] not (Filosofo4. comer and Filosofo5.comer) e foi traduzido para:“ A[] not (Filosofo5 . comem and Filosofo1 . comem)”:

Já o tradutor baseado em regras foi capaz de traduzir a saída correta. No entanto, o tradutor baseado em regras possui algumas limitações:

a) Os marcadores temporais precisam ser inseridos na estrutura dos requisitos para eles serem reconhecidos. No entanto, os marcadores temporais como: “sempre”, “nunca” são definidos como palavras ambíguas pelo padrão ISO/IEC/IEEE 29148-2018 (IEEE, 2018);

b) Não consegue reconhecer sentenças longas com mais de uma condição, como no exemplo: "REQUISITO": "Se a informação (genérico, nome de usuário e senha) digitada na janela de **login não é validado**, então o **aplicativo não deve ser inicializado**. O **botão OK não deve ser habilitado** a menos que os dados sejam validados em relação ao banco de dados.";

c) Não reconhece uma palavra composta da classe PROPN como sendo o mesmo processo, por exemplo: MS RSS para garantir que essa composição

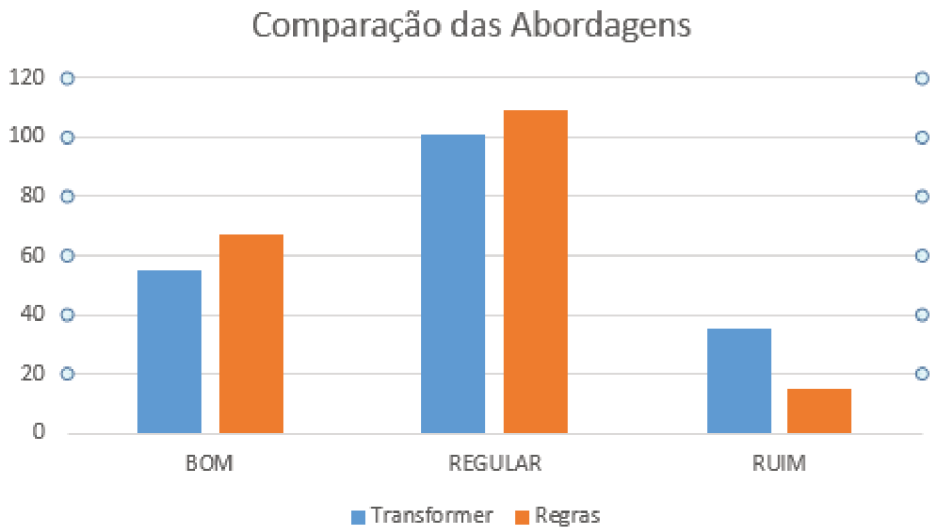
seja traduzida como um processo deve se juntar as palavras: MSRSS, caso contrário, as duas palavras serão reconhecidas como um processo individual: "MS"e "RSS"

d) Os processos devem ser definidos individualmente para manter a qualidade da tradução, por exemplo, na sentença: “O filósofo 1 e o filósofo 2 comeram” foram definidos “filósofo 1” e “filósofo 2”. Caso fosse definido: “O filósofo 1 e o 2” a frase seria suportada, mas não teria o mesmo sentido. O tradutor reconheceria “filósofo 1”, mas não reconheceria “filósofo 2”; e

e) Não faz a validação da sentença, e caso essa sentença esteja escrita fora das regras, não retorna nenhum tipo de erro.

O comparativo da classificação da tradução das abordagens pode ser observado na Figura 4.7

Figura 4.7 – Comparação das Abordagens



Fonte: Elaborado pelo Autor

No gráfico foi considerado o conjunto de dados contendo todos os requisitos de todos os casos de estudo. A rede LSTM não foi considerada por apresentar todas as transformações classificadas como “RUIM”. O tradutor baseado em regras apresenta uma melhor qualidade na tradução conforme a classificação realizada. A quantidade de traduções consideradas ruins também é menor para o tradutor baseado em regras. A Tabela 4.25 mostra os valores considerados para estruturação do gráfico da Figura 4.7.

Tabela 4.25 – Resultados Consolidados das Abordagens

Classe	Transformer	Regras
BOM	55	67
REGULAR	101	109
RUIM	35	15

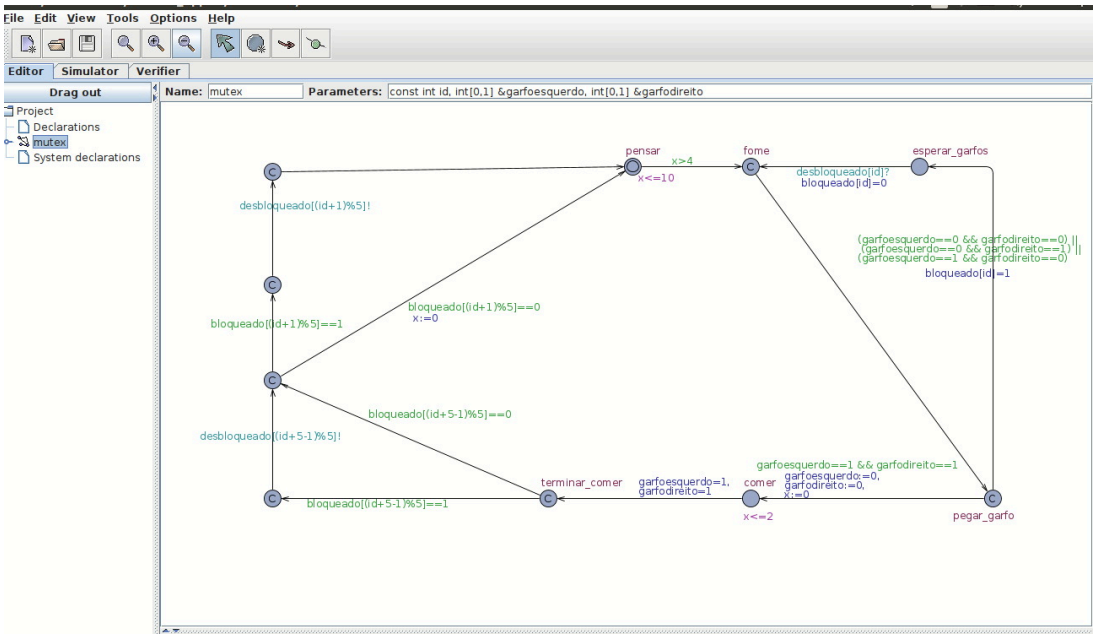
Fonte: Elaborado pelo Autor.

4.6 VERIFICAÇÃO DOS MODELOS NA FERRAMENTA UPPAAL

Os casos de estudo: jantar dos filósofos e protocolo de Fischer, foram modelados na ferramenta UPPAAL. As propriedades geradas foram utilizadas no processo de verificação dos modelos. Todas as propriedades geradas para o protocolo de Fischer (Figura 4.10) puderam ser verificados, no entanto, as propriedades de vivacidade referentes as seções críticas como: “processo1.req –> processo1.regiao_critica”, foram violadas (Esse resultado pode ser observado na Figura 4.11). Segundo Behrmann, David e Larsen (2004) a interpretação é que devido a modelagem, o processo pode ficar em espera para sempre, portanto impediria o acesso à seção crítica.

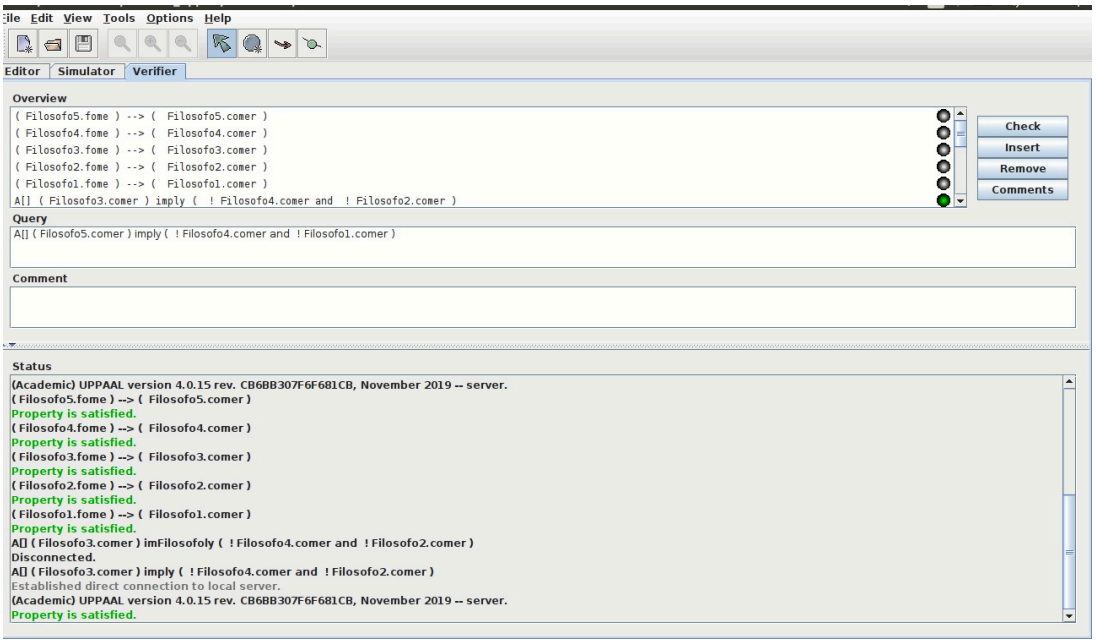
Já para o jantar dos filósofos (Figura 4.8) todas as propriedades foram satisfeitas no processo de verificação exibido pela Figura 4.9. As propriedades da base de Dwyer não foram verificadas pois essas propriedades não possuíam um modelo associado.

Figura 4.8 – Modelo Filósofos em UPPAAL



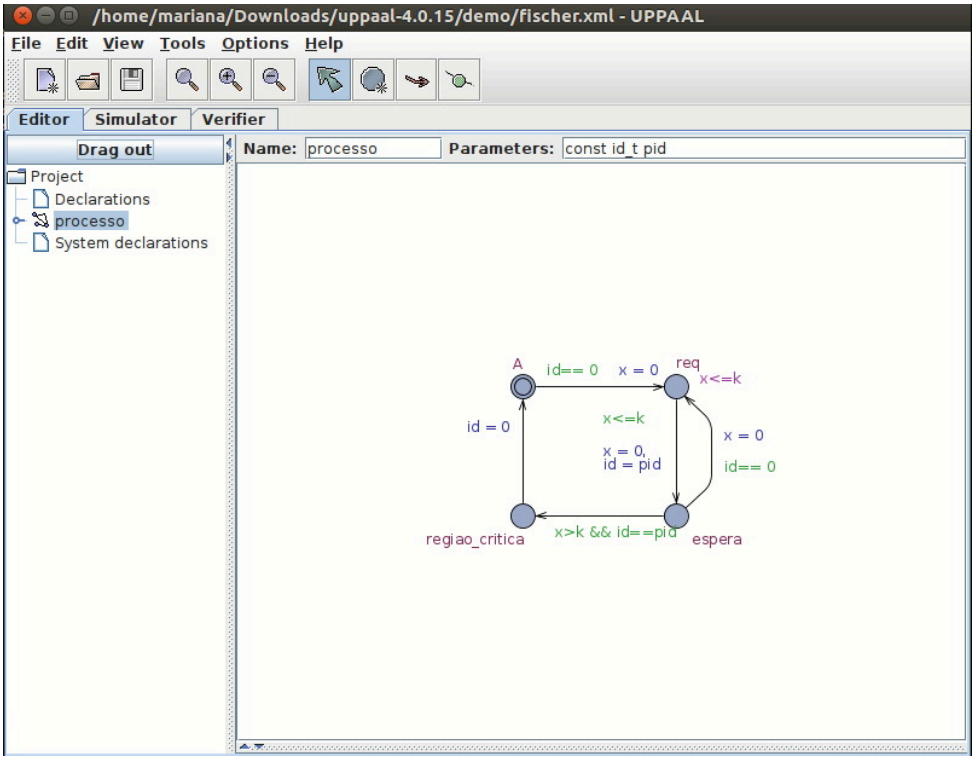
Fonte: Elaborado pelo Autor

Figura 4.9 – Modelo Filósofos em UPPAAL - Verificação de Propriedades



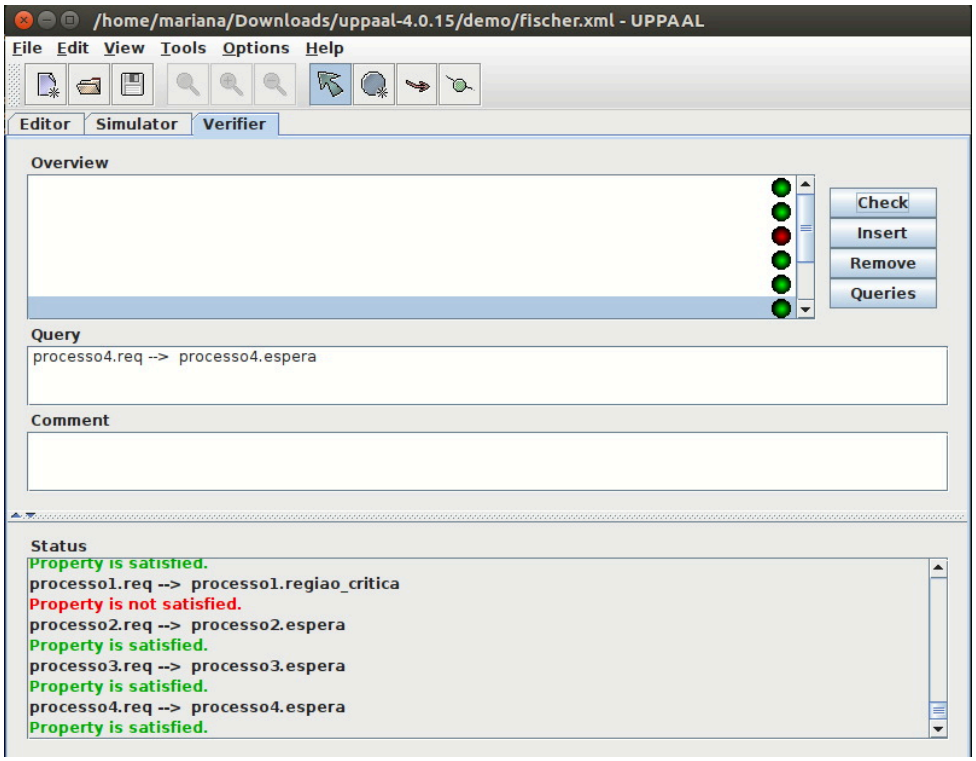
Fonte: Elaborado pelo Autor

Figura 4.10 – Modelo Protocolo Fischer em UPPAAL



Fonte: Adaptado de Behrmann, David e Larsen (2004).

Figura 4.11 – Modelo Protocolo Fischer em UPPAAL - Verificação de Propriedades



Fonte: Adaptado de Behrmann, David e Larsen (2004).

4.7 CONSIDERAÇÕES SOBRE OS MÉTODOS ADOTADOS

O tradutor baseado em regras conseguiu atingir os objetivos e atender os requisitos especificados, no entanto, esse tradutor é limitado visto que depende de uma formatação específica no requisito que será traduzido.

O tradutor baseado em rede neural apesar de não conseguir lidar com os diagramas e reconhecer sua estrutura, apresenta uma vantagem sobre a tradução baseada em regras que é a capacidade de lidar com todas as especificações para o qual ele for treinado, mesmo que as estruturas das sentenças sejam longas. Além disso, a rede neural não se limita a um controle de estilo de escrita.

Uma limitação na abordagem de tradução baseada em rede neural é a estruturação de uma base de treinos otimizada que contenha conjuntos de requisitos e propriedades com alta qualidade.

Apesar das abordagens se tratarem de uma tradução automática, ambas precisaram ter seus resultados verificados manualmente de modo a garantir a qualidade da tradução realizada.

4.8 CONSIDERAÇÕES FINAIS

Os resultados dos experimentos mostraram que a tradução de propriedades é possível utilizando às duas abordagens: tradução baseada em regras e a baseada em

redes neurais. No entanto, existem algumas limitações apresentadas na seção 4.7 que devem ser revisitadas.

Segundo os estudos apresentados no capítulo 2, o algoritmo de aprendizagem profunda apresenta bons resultados em atividades de tradução, e por esse motivo é interessante que ele seja mais explorado no âmbito de tradução de propriedades temporais. Necessita-se buscar na literatura, mais exemplos de requisitos verificáveis e suas expressões correspondentes em lógica temporal para aumentar e melhorar a base de dados utilizada para treino. Apesar das limitações relatadas na seção 4.7 e na seção 4.5.4, o tradutor baseado em regras conseguiu traduzir as propriedades especificadas nos casos de estudo Jantar dos filósofos e protocolo de Fischer e dessa forma os modelos puderam ser verificados.

5 CONCLUSÃO

Diante do crescente aumento da complexidade dos *sistemas de software* e a necessidade de que esses *sistemas de software* sejam o mais seguro e robusto possível, a verificação e validação se tornam cada vez mais fundamentais. A verificação de modelos permite encontrar inconsistências ainda no início do projeto. No entanto, a aplicação desse método depende da modelagem formal dos *sistemas de software*, que não é tão difundida na indústria quanto a modelagem semi-formal (como a UML).

Considerando esse cenário, o trabalho apresentou conceitos sobre a modelagem semi-formal UML e SysML, e sobre a modelagem formal de sistemas de software. Foram abordados conceitos do processo de verificação formal além de ferramentas e linguagens utilizadas para a aplicação da verificação.

O trabalho mostrou uma comparação entre algumas ferramentas e linguagens de verificação visando definir uma capaz de atender as necessidades de sistemas concorrentes e também de sistemas de tempo real visando os possíveis trabalhos futuros.

Adicionalmente o trabalho apresentou a tradução de requisitos escritos na linguagem natural em Português para propriedades temporais formais na linguagem UPPAAL. Na modelagem semi-formal as propriedades são geralmente encontradas em formas textuais escritas em linguagem natural. Por esse motivo foram apresentadas duas abordagens de processamento de linguagem natural: tradução baseada em regras e tradução baseada em redes neurais utilizando duas estruturas de rede: RNN-LSTM (Codificador-Decodificador) e *transformer*.

As duas abordagens foram utilizadas no processo de tradução de requisitos escritos em Português para lógica temporal. A rede *transformer* apresentou um melhor resultado comparada com a rede LSTM. Na rede *transformer*, cerca de 18% das traduções obtiveram um resultado muito diferente do resultado esperado com uma precisão inferior resultante de 0,5. Já a rede LSTM não obteve bons resultados na tradução.

O tradutor baseado em regras obteve um resultado satisfatório na tradução das propriedades apesar de restringir como os requisitos são escritos. Apesar dessas abordagens serem tratadas como tradução automática, ambas tiveram seus resultados pós processados manualmente de modo a garantir a qualidade da tradução.

Os tradutores foram utilizados na transformação de propriedades de segurança, ausência de impasse (*deadlock*) e vivacidade no problema do Jantar dos Filósofos, além das propriedades referentes ao protocolo Fischer. Os casos de estudo foram modelados manualmente em UPPAAL e as propriedades puderam ser verificadas.

As abordagens ainda apresentam limitações apresentadas na seção 4.7 e na seção 4.5.4. Portanto, considera-se que existem vários caminhos a serem percorridos

diante dos desafios apresentados pela tradução de propriedades.

Além das limitações já abordadas nesse trabalho, como: manter e alimentar uma base de dados representativa e com alta qualidade, trabalhar na abordagem neural para conseguir atender os demais requisitos estabelecidos na proposta de tradução, rever a gramática de entrada do tradutor baseado em regras para conseguir reconhecer o máximo de requisitos possíveis.

Assim, pode-se considerar que existem ainda outros desafios que podem ser trabalhados como: as traduções de propriedades quantitativas que não foram abordas e são de interesse na verificação de sistemas de tempo real.

Um desafio nessa área é reconhecer uma linguagem não restrita (ambígua e de gramática não controlada). Considerando o tradutor baseado em regras, esse desafio se torna complexo, visto que as construções de frases e sentenças na língua portuguesa são diversificadas. Assim, elaborar uma regra que contemple todas as possibilidades podem ser de um problema não trivial.

Como trabalho futuro além dos desafios já mencionados existe ainda uma segunda abordagem que complementa o estudo da tradução de propriedades: a transformação de modelos. Considerando que os modelos usuais expressados em UML ou SysML não permitem a aplicação do método de verificação formal, é de interesse que esses modelos sejam traduzidos para uma modelagem formal de maneira que seja possível realizar a verificação deles.

REFERÊNCIAS

Renesse, F.; Aghvami, A. H. Formal verification of ad-hoc routing protocols using spin model checker. In: *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference (IEEE Cat. No.04CH37521)*. [S.l.: s.n.], 2004. v. 3, p. 1177–1182 Vol.3.

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>.

ALVES, J. L. *REDES NEURAIS RECORRENTES APLICADAS À CLASSIFICAÇÃO DE FAKE NEWS EM LÍNGUA PORTUGUESA*. 2019. Online; accessed 10 Sep 2021. Disponível em: <app.uff.br/riuff/bitstream/1/13195/1/Dissertaç~ao_Jairo.pdf>.

ASD. *ASD Simplified Technical English, Specification ASD-STE100*. Brussels, Belgium, 2017.

B., M. A. R. D. A. Model-driven development within a legacy system: An industry experience report. In: *2005 Australian Software Engineering Conference*. [S.l.: s.n.], 2005. p. 14–22.

BAHDANAU, D.; CHO, K.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. *ArXiv*, v. 1409, 09 2014.

BAIER, C.; KATOEN, J.-P. *Principles of Model Checking (Representation and Mind Series)*. [S.l.]: The MIT Press, 2008. ISBN 026202649X, 9780262026499.

BEHRMANN, G.; DAVID, A.; LARSEN, K. A tutorial on uppaal. In: . [S.l.: s.n.], 2004. v. 3185, p. 200–236.

BERTHOMIEU, B.; RIBET, P.-O.; VERNADAT, F. The tool tina – construction of abstract state spaces for petri nets and time petri nets. *International Journal of Production Research*, Taylor & Francis, v. 42, n. 14, p. 2741–2756, 2004. Disponível em: <https://doi.org/10.1080/00207540412331312688>.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: guia do usuário*. 1a edição. ed. [S.l.]: Elsevier Brasil, 2012.

BOURQUE, P.; RICHARD, E. *Swebok Version 3.0 - Guide to the Software Engineering Body of Knowledge*. [S.l.]: IEEE, 2014. ISBN 10: 0-7695-5166-1.

BRAGA, A. de P. *Redes neurais artificiais: teoria e aplicações*. LTC Editora, 2007. ISBN 9788521615644. Disponível em: <https://books.google.com.br/books?id=R-p1GwAACAAJ>.

C. CUNHA G. B., F. P. R. B. *Lógica matemática*. [S.l.]: Núcleo de Tecnologia Educacional da Universidade Federal de Santa Maria, 2017. ISBN 978-85-8341-184-0.

CAN, P. et al. Modeling and verification of can bus with application layer using uppaal. *Electronic Notes in Theoretical Computer Science*, Elsevier B.V., v. 309, n. 3, p. 31–49, 2014.

Cavada R. et al. *NuSMV 2.6 Tutorial*. [S.l.], 2015.

CHEN, M. Translating natural language into linear temporal logic. In: . [S.l.: s.n.], 2018.

CIMATTI, A. et al. Nusmv 2: An opensource tool for symbolic model checking. *Lecture Notes in Computer Science*, Computer Aided Verification, v. 2404, n. 3, p. 359–364, 2002.

CLARKE, E. e. a. Bounded model checking using satisfiability solving. formal methods in system design. Kluwer Academic Publishers, v. 19, n. 1, p. 7–34, ago. 2004.

D., M. J. S. S. K. U. A. W. *MDA Distilled: Principles of Model-Driven Architecture*. 1. ed. [S.l.]: Addison-Wesley Professional, 2004. ISBN 10 : 0201788918.

DISBORG, K. *Advantages and disadvantages with Simplified Technical English : to be used in technical documentation by Swedish export companies*. Dissertação (Mestrado) — Linköping University, 01 2007.

DUBY, C. K. *Accelerating Embedded Software Development with a Model Driven Architecture ®*. [S.l.], 2003.

DULAC, N. et al. Using system dynamics for safety and risk management in complex engineering systems. In: *Proceedings of the Winter Simulation Conference, 2005.* [S.l.: s.n.], 2005. p. 10 pp.–. ISSN 0891-7736.

DWYER, M. B.; AVRUNIN, G. S.; CORBETT, J. C. Patterns in property specifications for finite-state verification. In: *Proceedings of the 21st International Conference on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 1999. (ICSE '99), p. 411–420. ISBN 1581130740. Disponível em: <https://doi.org/10.1145/302405.302672>.

Dwyer, M. B.; Avrunin, G. S.; Corbett, J. C. Patterns in property specifications for finite-state verification. In: *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)*. [S.l.: s.n.], 1999. p. 411–420.

Dzifcak, J. et al. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In: *2009 IEEE International Conference on Robotics and Automation*. [S.l.: s.n.], 2009. p. 4163–4168.

ELMAN, J. L. Finding structure in time. *Cognitive Science*, v. 14, n. 2, p. 179 – 211, 1990. ISSN 0364-0213. Disponível em: <http://www.sciencedirect.com/science/article/pii/036402139090002E>.

EMERSON, E. A. Chapter 16 - temporal and modal logic. In: VAN LEEUWEN, J. (Ed.). *Formal Models and Semantics*. Amsterdam: Elsevier, 1990, (Handbook of Theoretical Computer Science). p. 995 – 1072. ISBN 978-0-444-88074-1. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780444880741500214>.

Farn Wang. Formal verification of timed systems: a survey and perspective. *Proceedings of the IEEE*, v. 92, n. 8, p. 1283–1305, 2004.

France, R.; Rumpe, B. Model-driven development of complex software: A research roadmap. In: *Future of Software Engineering (FOSE '07)*. [S.l.: s.n.], 2007. p. 37–54.

GARDEY, G. et al. Romeo: A tool for analyzing time petri nets. 2005.

GESSER, C.; FURTADO, O. Gals : Gerador de analisadores léxicos e sintáticos. 12 2002.

GHERBI, A.; KHENDEK, F. Uml profiles for real-time systems and their applications. *Journal of Object Technology*, v. 5, p. 149–169, 05 2006.

GHOMARI, A.; DJERABA, C. Modeling multimedia synchronization using a time petri net based approach. In: ____. [S.l.: s.n.], 2010. ISBN 978-953-307-108-4.

GHOSH, S. et al. Arsenal: Automatic requirements specification extraction from natural language. In: RAYADURGAM, S.; TKACHUK, O. (Ed.). *NASA Formal Methods*. Cham: Springer International Publishing, 2016. p. 41–46. ISBN 978-3-319-40648-0.

GRAVES, A. Long short-term memory. In: ____. [S.l.: s.n.], 2012. p. 37–45. ISBN 978-3-642-24796-5.

Harris, C. B.; Harris, I. G. Generating formal hardware verification properties from natural language documentation. In: *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*. [S.l.: s.n.], 2015. p. 49–56.

HAYKIN, S. *Redes Neurais: Princípios e Prática*. Artmed, 2007. ISBN 9788577800865. Disponível em: <https://books.google.com.br/books?id=bhMwDwAAQBAJ>.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Computation*, v. 9, p. 1735–1780, 1997.

HOLZMANN, G. J. *Design and Validation of Computer Protocols*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1991. ISBN 0-13-539925-4.

IEEE. Ieee standard classification for software anomalies. *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)*, p. 1–23, 2010.

IEEE. Iso/iec/ieee international standard - systems and software engineering - life cycle management – part 1:guidelines for life cycle management. *ISO/IEC/IEEE 24748-1:2018(E)*, p. 1–82, 2018.

KALCHBRENNER, N.; BLUNSOM, P. Recurrent continuous translation models. *EMNLP 2013 - 2013 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, v. 3, p. 1700–1709, 01 2013.

KHURANA, D. et al. Natural language processing: State of the art, current trends and challenges. 08 2017.

KLEPPE A.AND WARMER, J.; BAST, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. 1. ed. [S.l.]: Addison-Wesley Professional, 2003.

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. v. 14, 03 2001.

LAMPORT, L. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002. Disponível em: <https://www.microsoft.com/en-us/research/publication/specifying-systems-the-tla-language-and-tools-for-hardware-and-software-engineers/>.

LECOMTE, T. et al. Applying a formal method in industry: A 25-year trajectory. In: . [S.l.: s.n.], 2017. p. 70–87. ISBN 978-3-319-70847-8.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, p. 436–44, 05 2015.

LENNY, D. *SysML Distilled: A Brief Guide to the Systems Modeling Language*. [S.l.]: Addison-Wesley Professional, 2014. ISBN 978-0-321-92786-6.

LEVESON, N. G.; TURNER, C. S. An investigation of the therac-25 accidents. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 26, n. 7, p. 18–41, jul. 1993. ISSN 0018-9162. Disponível em: <http://dx.doi.org/10.1109/MC.1993.274940>.

LEWIS, M. *NASA Shares Initial Findings from Boeing Starliner Orbital Flight Test Investigation*. 2020. [Online; accessed 21-Octuber-2020]. Disponível em: <https://blogs.nasa.gov/commercialcrew/2020/02/07/nasa-shares-initial-findings-from-boeing-starliner-orbital-flight-test-investigation/>.

LIKER, J. K.; OGDEN, T. N. *A Crise da Toyota: Como a Toyota Enfrentou o Desafio dos Recalls e da Recessão para Ressurgir Mais Forte*. 1a edição. ed. [S.l.]: Bookman, 2012.

LIME, D. et al. Romeo: A parametric model-checker for Petri nets with stopwatches. In: KOWALEWSKI, S.; PHILIPPOU, A. (Ed.). *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*. York, United Kingdom: Springer, 2009. (Lecture Notes in Computer Science, v. 5505), p. 54–57.

Lovengreen H. [S.l.], 2016.

MAZIERO, E. G.; PARDO, T. A. S.; NUNES, M. d. G. V. *Identificação automática de segmentos discursivos: o uso do parser PALAVRAS.* 2016. Disponível em: <http://repositorio.icmc.usp.br/handle/RIICMC/6726?show=full>.

Merlin, P.; Farber, D. Recoverability of communication protocols - implications of a theoretical study. *IEEE Transactions on Communications*, v. 24, n. 9, p. 1036–1043, 1976.

MICROSOFT. *TLA+ Proof System*. [S.l.], 2020. Disponível em: <https://tla.msr-inria.inria.fr/tlaps/content/Home.html>.

MIKHAIL, E.; ACKERMANN, F. *Observations and Least Squares*. IEP, 1976. (The IEP series in civil engineering). ISBN 9780700224814. Disponível em: <https://books.google.com.br/books?id=Yx3vAAAAMAAJ>.

MILLER, S. P.; WHALEN, M. W.; COFER, D. D. Software model checking takes off. *Commun. ACM*, ACM, New York, NY, USA, v. 53, n. 2, p. 58–64, fev. 2010. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1646353.1646372>.

MORRISSEY, S. Data-driven machine translation for sign languages. In: . [S.l.: s.n.], 2008.

NELKEN, R.; FRANCEZ, N. Automatic translation of natural language system specifications. In: . [S.l.: s.n.], 1996. p. 360–371. ISBN 978-3-540-61474-6.

NIRENBURG, S.; WILKS, Y. Machine translation. In: ZELKOWITZ, M. V. (Ed.). *Fortieth Anniversary Volume: Advancing into the 21st Century*. Elsevier, 2000, (Advances in Computers, v. 52). p. 159–188. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0065245800800182>.

OMG. *WHAT IS SYSML?* 2007. Online; accessed 15 Aug 2021. Disponível em: <https://www.omgsysml.org/what-is-sysml.htm>.

OMG. *UML 2.0 Infrastructure Specification.* [S.l.], 2011.

OSSADA, J.; MARTINS, L. E. Gerse: Guia para elicitação de requisitos de sistemas embarcados. In: *Anais do WER12 – Workshop em Engenharia de Requisitos, Buenos Aires, Argentina, Abril 24–27, 2012*. [S.l.: s.n.], 2012. p. 1–14.

OTHERO, G. Lingüística computacional: uma breve introdução. *Letras de Hoje*, v. 41, 09 2006.

PETRE, M. Uml in practice. *2013 35th International Conference on Software Engineering (ICSE)*, p. 722–731, 2013.

PRESSMAN, R. S. *Engenharia de software.* [S.l.]: MAKRON Books, 1995. ISBN 10: 8534602379.

PÁDUA, W. d. *Engenharia de software: Fundamentos, métodos e padrões.* [S.l.]: LTC, 2011.

RAMADHAN, Z.; SIAHAAN, A. P. U. Dining philosophers theory and concept in operating system scheduling. *IOSR Journal of Computer Engineering*, v. 18, p. 2278–661, 12 2016.

RAMAN, V. et al. Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using natural language. In: *Robotics: Science and Systems*. [S.l.: s.n.], 2013.

RAMCHANDANI, C. *ANALYSIS OF ASYNCHRONOUS CONCURRENT SYSTEMS BY TIMED PETRI NETS*. USA, 1974.

ROCHE, S. *Finite-State Language Processing*. [S.l.]: The MIT Press, 1997. ISBN 9780262290951.

ROZIER, K. Linear temporal logic symbolic model checking. *Computer Science Review*, v. 5, p. 163–203, 05 2011.

Schmidt, D. C. Guest editor's introduction: Model-driven engineering. *Computer*, v. 39, n. 2, p. 25–31, 2006.

SERUGENDO, G. D. M. et al. Real-time synchronised petri nets. In: . [S.l.: s.n.], 2002. v. 2360, p. 142–162.

SINGH, B. et al. A trade-off between ML and DL techniques in natural language processing. *Journal of Physics: Conference Series*, IOP Publishing, v. 1831, n. 1, p. 012025, mar 2021. Disponível em: <https://doi.org/10.1088/1742-6596/1831/1/012025>.

SOBEIH, A.; VISWANATHAN, M.; HOU, J. Check and simulate: a case for incorporating model checking in network simulation. In: . [S.l.: s.n.], 2004. p. 27 – 36. ISBN 0-7803-8509-8.

SOMMERVILLE, I. *Software Engineering*. [S.l.]: Pearson/Addison-Wesley, 2004. (International computer science series). ISBN 9780321210265.

STURLA; GIANCARLO. A two-phased approach for natural language parsing into formal logic. In: . [S.l.: s.n.], 2017.

STöRRLE, H. How are conceptual models used in industrial software development?: A descriptive survey. In: . [S.l.: s.n.], 2017. p. 160–169.

SUTSKEVER, I.; VINYALS, O.; LE, Q. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, v. 4, 09 2014.

SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. Disponível em: <http://arxiv.org/abs/1409.3215>.

TASKIN, Z.; AL, U. *Natural Language Processing Applications in Library and Information Science*. 2019.

TAURION, C. *Softwares Embarcados – A nova onda da Informática chips e softwares em todos objetos 1ªedição*. Rio de Janeiro: Brasport, 2005. ISBN 026202649X, 9780262026499.

UPPAAL. *Uppaal4.0 : Small Tutorial*. [S.l.], 2009.

VARSAMOPOULOS, S.; BERTELS, K.; ALMUDEVER, C. G. Comparing neural network based decoders for the surface code. *IEEE Transactions on Computers*, Institute of Electrical and Electronics Engineers (IEEE), v. 69, n. 2, p. 300–311, Feb 2020. ISSN 2326-3814. Disponível em: <http://dx.doi.org/10.1109/TC.2019.2948612>.

VASWANI, A. et al. *Attention Is All You Need*. 2017.

VEREIJKEN, J. J. Fischer's protocol in timed process algebra. 01 1996.

WANG, C. et al. *Learning a natural-language to LTL executable semantic parser for grounded robotics*. 2021.

WANG, F. Formal verification of timed systems: A survey and perspective. *Commun. ACM*, IEEE, v. 92, n. 8, p. 1283–1305, ago. 2004.

WAYNE, H. *Practical TLA+ Planning Driven Development*. [S.l.]: Apress, 2018.

WU, Y. et al. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016.

Yan, R.; Cheng, C.; Chai, Y. Formal consistency checking over specifications in natural languages. In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. [S.l.: s.n.], 2015. p. 1677–1682.

YANG, S.; WANG, Y.; CHU, X. *A Survey of Deep Learning Techniques for Neural Machine Translation*. 2020.

YANG, S.; WANG, Y.; CHU, X. A survey of deep learning techniques for neural machine translation. *ArXiv*, abs/2002.07526, 2020.

YIN, W. et al. Comparative study of cnn and rnn for natural language processing. 02 2017.

YOUNG, T. et al. Recent trends in deep learning based natural language processing. *CoRR*, abs/1708.02709, 2017. Disponível em: <http://arxiv.org/abs/1708.02709>.

YU, Y.; MANOLIOS, P.; LAMPORT, L. Model checking tla+ specifications. In: *Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*. London, UK, UK: Springer-Verlag, 1999. (CHARME '99), p. 54–66. ISBN 3-540-66559-5. Disponível em: <http://dl.acm.org/citation.cfm?id=646704.702012>.

ZHONGSHENG, Q.; XIN, L.; XIAOJIN, W. Modeling distributed real-time elevator system by three model checkers. *International Journal of Online Engineering (iJOE)*, v. 14, p. 94, 04 2018.

APÊNDICE A – PROCESSOS DE ENGENHARIA DE SOFTWARE

A.1 ESPECIFICAÇÃO DE REQUISITOS

A especificação de requisitos é uma das etapas fundamentais para construção de um software. Nessa etapa são definidas as principais características e limitações do sistema como um todo. A partir dos requisitos é possível compreender a complexidade do sistema que se deseja construir, além de definir os objetivos que esse sistema precisa atender. Segundo (SOMMERVILLE, 2004), os requisitos definem: funções, propriedades essenciais e desejáveis e também as limitações do software.

No entanto os requisitos podem apresentar inconsistências, e conflitar entre si, podendo causar problemas no produto final. Muitos erros de projetos são provenientes de requisitos mal escritos e consequentemente mal interpretados (PRESSMAN, 1995; PÁDUA, 2011).

Um forma de validar esses requisitos é através da modelagem do comportamento do sistema. A modelagem permite visualizar as especificações do sistema em alto nível, facilitando a comunicação entre os envolvidos no projeto sobre os aspectos e requisitos fundamentais do sistema.

A.2 DESENVOLVIMENTO POR MODELO

A.2.1 Model Driven Engineering - MDE

A abordagem *Model Driven Engineering* (MDE) traz uma ideia de que a bordagem orientada a modelos vai além da etapa de desenvolvimento, englobando todo o processo de engenharia de Software (Schmidt, 2006). No conceito do *Model Driven Engineering* (MDE), a modelagem é uma das etapas principais. A equipe modela o software, depois os modelos são interpretados e traduzidos, e em seguida são gerados códigos a partir do modelo.

O MDE ganhou força com a evolução das linguagens de programação que antes, eram limitadores para a utilização do conceito. Atualmente as linguagens se aproximam cada vez mais da linguagem natural, além disso a utilização de orientação a objetos e reuso, tornou esse conceito ainda mais atingível. O MDE tem como objetivo a modelagem de sistemas em um nível mais abstrato de forma a expressar o que o sistema faz e não como ele faz, ou seja o sistema é desenvolvido em alto nível (Schmidt, 2006).

O MDE engloba os conceitos de *Model Driven Development (MDD)* e *Model Driven Architecture (MDA)*, que serão apresentados nas próximas seções, fornecendo suporte para análises e decisões no desenvolvimento orientado a modelo (Schmidt, 2006).

A.2.2 Model Driven Development - MDD

O *Model Driven Development* (MDD) é uma abordagem que visa o desenvolvimento de software focado principalmente na modelagem (KLEPPE A.AND WARMER; BAST, 2003). Ao invés de programar um software em uma linguagem de programação,o software será modelado e então, o código será gerado a partir desse modelo. Essa abordagem torna o nível de abstração ainda mais elevado e independente de tecnologia, permitindo uma maior reusabilidade (D., 2004). Além disso, o MDD visa agilizar o desenvolvimento do software e também torná-lo mais econômico (B., 2005).

A abordagem Model-Driven Architecture (MDA) do Object Management Group (OMG) é um exemplo conhecido de MDD. Essa abordagem tem sido referida como uma modelagem de sistemas de software (DUBY, 2003)

A.2.3 Model Driven Architecture - MDA

Model Driven Architecture (MDA) é uma especificação que tem como objetivo encontrar formas de como um modelo pode ser transformado em código de forma automática. Esse conceito foi desenvolvido pela OMG e propõe uma padronização de abordagens orientadas a modelos (France; Rumpe, 2007). As especificações são disponibilizadas para uso de interessados em aplicar os conceitos do MDA. O MDA apresenta três conceitos sobre a modelagem de sistemas (France; Rumpe, 2007):

- O ponto de vista da independência de computação(CIM), que aborda o ambiente em que o sistema irá operar, e também dos recursos requeridos do sistema. O MDA aborda modelos independentes de computação .

- O ponto de vista independente de plataforma está interessado nos recursos que provavelmente não mudarão quando o sistema for usado em plataformas diferentes. Este ponto de vista produz modelos independentes de plataforma (PIM). Um modelo independente de plataforma é um modelo de sistema que não possui nenhuma informação de implementação específica de tecnologia.

- Um PIM é transformado em um ou mais modelos especifico de plataforma (PSM - *Platform Specific Model*) adicionando informações que o tornam específico para uma ou mais plataformas específicas. O PIM e o PSM especificam o mesmo sistema, mas o PSM é restrito a uma tecnologia específica e pode conter elementos específicos da plataforma. Duby (2003) destaca que o MDA permite uma reação rápida às mudanças nos requisitos funcionais e tecnológicos da plataforma, além de possibilitar uma reutilização em larga escala de modelos independentes de plataforma. Outras vantagens são a facilidade de documentação, custos reduzidos de garantia de qualidade e melhoria na qualidade do sistema.

No MDA ou em qualquer abordagem MDD, são utilizados modelos para a definir os sistemas de software. O MDA permite outras linguagens além da *Unified Modeling Language (UML)*, no entanto a UML é a linguagem padrão utilizada pela abordagem. O OMG vem trabalhando na evolução da UML. Segundo a OMG (2011), as atualizações para a UML 2.0 permitem que essa versão desempenhe um importante papel na abordagem MDA. Isso acontece porque a UML 2.0 suporta os conceitos mais proeminentes na visão em evolução do MDA (OMG, 2011) :

- Família de linguagens: UML é uma linguagem de propósito geral, que pode ser customizada para uma ampla variedade de domínios, plataformas e métodos. Nela pode-se personalizar dialetos UML para vários domínios (por exemplo, finanças, telecomunicações, aeroespacial), plataformas (por exemplo, J2EE, .NET) e métodos (por exemplo, Processo Unificado, métodos Ágeis).

- Especificando um sistema independentemente da plataforma que o suporta: UML 2.0, de propósito geral se destina a ser usada com uma ampla variedade de métodos de software. Consequentemente, inclui suporte para métodos de software que distinguem entre modelos de análise ou lógicos e modelos de design ou físicos. Como os modelos de análise ou lógicos são tipicamente independentes da implementação e das especificações da plataforma, eles podem ser considerados “Modelos Independentes de Plataforma” (PIMs), consistentes com a terminologia MDA em evolução.

- Especificando plataformas: O mecanismo de Perfil permite que os modeladores personalizem UML com mais eficiência para plataformas de destino, como J2EE ou .NET. Além disso, as construções para especificar arquiteturas de componentes, contêineres de componentes (ambientes de execução) são significativamente aprimorados, permitindo modeladores para especificar totalmente os ambientes de implementação de destino.

- Transformar a especificação do sistema em uma para uma plataforma particular: Refere-se à transformação de um modelo independente de plataforma em um modelo específico de plataforma. A UML especifica vários relacionamentos que podem ser usados para especificar a transformação de um PIM em um PSM.

APÊNDICE B – LÓGICAS TEMPORAIS - CONCEITOS

B.1 LÓGICA MODAL

A lógica de primeira ordem é limitada e não pode expressar os conceitos de possível e necessário. Através da lógica modal a representação desses conceitos é possível. O conceito de lógica modal é fundamental para o entendimento de lógica temporal, visto que essa possui operadores da lógica modal (C. CUNHA G. B., 2017).

A sintaxe e semântica da lógica modal se baseia na lógica de primeira ordem.

A seguir é são apresentados alguns exemplos que utilizam os quantificadores da lógica modal (C. CUNHA G. B., 2017):

- $\Diamond\Phi$: É possível que Φ seja verdade.
- $\Box\Phi$: É necessário que Φ seja verdade.
- $\Box\Phi \rightarrow \Diamond\Phi$: Tudo que é necessário é possível.
- $\Phi \rightarrow \Diamond\Phi$: Se algo é verdadeiro então é possível.
- $\Phi \rightarrow \Box\Phi$: Se algo é verdadeiro então é necessariamente possível.
- $\Phi \rightarrow \Box\Diamond\Phi$: Algo que é verdadeiro é necessariamente possível.
- ’ $\Phi \rightarrow$ "’Φ : Tudo que é possível necessariamente possível.

Nos exemplos apresentados, o conceito de necessidade é representado por $\Box$, já o conceito de Possibilidade é representado por $\Diamond$.

B.2 SEMÂNTICA DE KRIPKE

A semântica de Kripke descreve os modelos de Kripke, onde se tem um sistema k que representa o menor sistema modal, representando a intersecção de todos os sistemas modais, onde um conjunto de axiomas e regras de inferência são utilizadas para a sua representação formal.

P é definido como um conjunto de proposições atômicas e uma estrutura de Kripke é representada por uma tupla (S, i, R, L) (C. CUNHA G. B., 2017):

- S é definido pelo conjunto finito de estados.
- i $\in$ S é o estado inicial.
- R $\subseteq$ S $\times$ S é uma relação de transição total: $\forall s \in S . \exists s' \in S \cdot (s, s') \in R$
- L:S $\rightarrow$(P) é uma função que marca cada estado com o conjunto de fórmulas atômicas validas nesse estado.

A estrutura de Kripke pode ser utilizada para definir a semântica da lógica temporal. Técnicas de verificação de propriedades e especificação podem ser apresentadas independente do formalismo do modelo a ser adotado.

APÊNDICE C – PORTUGUÊS SIMPLIFICADO PARA A ESPECIFICAÇÃO DE REQUISITOS

Para o processo de tradução de modelos semi formais para modelos formais, é necessário encontrar no modelo semi formal, as especificações que serão traduzidas em propriedades e então verificadas pelo verificador de modelo. As especificações expressam as características do modelo, ou seja, são os requisitos fundamentais do sistema modelado que geralmente são descritos em linguagem natural.

Os requisitos devem ser claros, livres de contradições, portanto, não devem ser utilizados termos ambíguos em sua especificação.

Ao se escrever requisitos, os termos vagos e gerais devem ser evitados. A utilização desses termos podem gerar requisitos não verificáveis. Além disso, a utilização de termos vagos, pode gerar requisitos ambíguos. Um requisito ambíguo permite múltiplas interpretações (IEEE, 2018).

O padrão ISO/IEC/IEEE 29148-2018 (IEEE, 2018) fornece uma lista de termos considerados vagos:

- superlativos : melhor, mais;
- linguagem subjetiva: amigável, fácil de usar, econômico;
- pronomes vagos: ele, isto, aquilo;
- termos que são ambíguos como advérbios e adjetivos: quase sempre, significativa, mínimo;
- termos não verificáveis: fornece suporte, mas não limitado para;
- frases comparativas: melhor que, maior qualidade;
- brechas como: se possível, conforme apropriado, como aplicável;
- termos que sugerem totalidade: todos, sempre, tudo.

A língua portuguesa é naturalmente ambígua. Uma palavra ou até mesmo uma frase, pode ser interpretada de diversas formas dependendo do contexto. Todas essas regras e formas de escrita acabam dificultando a aprendizagem da língua portuguesa, principalmente em relação à interpretação de texto.

Essas características dificultam a aprendizagem da língua portuguesa, impactando diretamente na qualidade da especificação de requisitos escritos nessa linguagem.

C.0.1 Dificuldades encontradas ao Especificar Requisitos

Uma pesquisa realizada em 2009 (OSSADA; MARTINS, 2012) foi aplicada a 53 profissionais da área de Sistemas Embarcados de empresas localizadas em São Paulo. Os profissionais foram questionados sobre quais eram as principais dificuldades encontradas durante a análise de requisitos, e entre as respostas estavam: requisitos confusos; e requisitos ambíguos (OSSADA; MARTINS, 2012).

Em 2016, uma pesquisa (SUTSKEVER; VINYALS; LE, 2014b) aplicada em 10 países incluindo o Brasil, coletou dados sobre os principais problemas enfrentados na etapa de especificação. Nessa pesquisa foram entrevistadas 228 empresas. Dentre os resultados citados estão: requisitos incompletos; requisitos especificados de forma abstrata; e requisitos inconsistentes.

Vale-se, então, questionar quais são as possíveis causas que contribuem para os resultados apresentados nas pequisas. Nesse contexto, é possível perceber que muitos profissionais têm dificuldades com a forma de escrita do requisito. Essa dificuldade ao se escrever corretamente frases e sentenças, compromete a qualidade da informação resultando em requisitos ambíguos, vagos, portanto não verificáveis.

C.0.2 Inglês Simplificado

Em 1983, a *European Association of Aeroespace Industries* (ASD) começou uma investigação de como simplificar a forma de escrita de documentos na língua inglesa (ASD, 2017).

O resultado da pesquisa foi uma especificação de Inglês Simplificado, o ASD-STE100. O ASD-STE100 surgiu com a finalidade de ajudar que pessoas envolvidas em um projeto fossem capazes de ler a documentação, principalmente pessoas que não tinham o inglês como língua materna. No entanto, o sucesso foi tanto, que outras indústrias o utilizam além do objetivo pretendido da documentação de manutenção da aviação (ASD, 2017).

O ASD-STE100 tem como propósito guiar a escrita de textos técnicos de forma clara, simples e não ambígua, utilizando palavras de fácil entendimento. O Inglês Simplificado fornece um conjunto de regras de escrita e um dicionário de vocabulário controlado. As regras de escrita definem aspectos de gramática e o dicionário especifica palavras gerais que podem ser utilizadas. As palavras foram escolhidas por sua simplicidade e facilidade de reconhecimento (ASD, 2017).

O trabalho apresentado em (DISBORG, 2007) concluiu que o uso do Inglês Simplificado resulta em uma documentação mais uniforme, independentemente do número de escritores envolvidos no projeto. Além disso, o ASD-STE100 se mostrou útil para empresas em que escritores técnicos não nativos em inglês, escrevem documentação.

Considerando esse cenário, a próxima seção tem como objetivo propor a criação

de um documento de Português Simplificado.

C.0.3 Português Simplificado

A criação de um documento definindo um subconjunto do Português, segue o propósito de atenuar o surgimento de informações ambíguas e vagas.O conjunto é constituído pelas seguintes regras:

- O operador ”e” só poderá ser utilizado para expressar uma condição que ocorre em paralelo, esse operador não poderá ser utilizado para expressar ações sequenciais, por exemplo:

  O avião decola e pousa.

  Esse exemplo define uma atividade sequencial que quer dizer que o avião decola e em algum momento depois ele pousa. As duas ações, pousar e decolar, não ocorrem ao mesmo tempo. Na expressão :

  O motorista dirige e ouve música

  o operador ”e” pode ser utilizado pois expressa que as ações estão sendo realizadas em conjunto.

- As formas nominais (infinitivo, gerúndio e particípio) não podem ser utilizadas nas sentenças, pois pode gerar ambiguidade. Por exemplo:

  o botão desabilita o campo enviando o comando.

  Na sentença não é possível definir quem envia o comando, se o botão para desabilitar o campo, ou se o campo estava enviando um comando ao ser desabilitado.

- Termos que expressam totalidade devem ser utilizados associados a condições. Por exemplo:

  *P*edro sempre escova os dentes.

  A sentença acima, não é permitida pois poderia ser interpretada de forma errônea. Poderia parecer que Pedro executa a ação de escovar os dentes 24 horas por dia sem qualquer intervalo e por tempo indefinido. No entanto, a sentença abaixo é permitida:

  *S*empre que o botão start for pressionado, a tela de inicialização deverá ser exibida.

- os termos comparativos permitidos são:
  - maior
  - maior ou igual

  - menor
  - menor ou igual
  - igual

- Termos subjetivos não devem ser utilizados. Por exemplo:

  O carro deve ser veloz.

  Não é possível, definir o significado de veloz na sentença acima. Um carro convencional pode andar a 100 km/h e ser considerado veloz, no entanto um carro de fórmula um que só alcança uma velocidade de 100 km/h não será considerado veloz.