UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

THIAGO ALVES ROCHA

COMPLEXIDADE DESCRITIVA DE CLASSES PROBABILÍSTICAS DE TEMPO POLINOMIAL E DAS CLASSES $\oplus P$ E $NP \cap coNP$ ATRAVÉS DE LÓGICAS COM QUANTIFICADORES GENERALIZADOS DE SEGUNDA ORDEM

FORTALEZA, CEARÁ

2014

THIAGO ALVES ROCHA

COMPLEXIDADE DESCRITIVA DE CLASSES PROBABILÍSTICAS DE TEMPO POLINOMIAL E DAS CLASSES $\oplus P$ E $NP \cap coNP$ ATRAVÉS DE LÓGICAS COM QUANTIFICADORES GENERALIZADOS DE SEGUNDA ORDEM

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Lógica e Inteligência Artificial

Orientador: Profa. Dra. Ana Teresa de Castro Martins

FORTALEZA, CEARÁ

2014

A000z ROCHA, T. A..

Complexidade Descritiva de Classes Probabilísticas de Tempo Polinomial e das Classes $\oplus P$ e $NP \cap coNP$ Através de Lógicas com Quantificadores Generalizados de Segunda Ordem / Thiago Alves Rocha. 2014.

81p.;il. color. enc.

Orientador: Profa. Dra. Ana Teresa de Castro Martins

Co-Orientador:

Dissertação(Ciência da Computação) - Universidade Federal do Ceará, Departamento de Computação, Fortaleza, 2014.

1. Complexidade Descritiva 2. Classes de Complexidade Probabilísticas 3. Quantificadores Generalizados I. Profa. Dra. Ana Teresa de Castro Martins(Orient.) II. Universidade Federal do Ceará– Ciência da Computação(Mestrado) III. Mestre

CDD:000.0

THIAGO ALVES ROCHA

COMPLEXIDADE DESCRITIVA DE CLASSES PROBABILÍSTICAS DE TEMPO POLINOMIAL E DAS CLASSES $\oplus P$ E $NP \cap coNP$ ATRAVÉS DE LÓGICAS COM QUANTIFICADORES GENERALIZADOS DE SEGUNDA ORDEM

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Área de concentração: Lógica e Inteligência Artificial

Aprovada em: __/__/____

BANCA EXAMINADORA

Profa. Dra. Ana Teresa de Castro Martins
Universidade Federal do Ceará - UFC
Orientadora

Prof. Dr. João Fernando Lima Alcântara
Universidade Federal do Ceará - UFC

Prof. Dr. Carlos Eduardo Fisch de Brito
Universidade Federal do Ceará - UFC

Prof. Dr. Ruy José Guerra Barretto de Queiroz
Universidade Federal de Pernambuco - UFPE

RESUMO

Vários problemas computáveis podem ser resolvidos de maneira mais eficiente ou mais natural através de algoritmos probabilísticos, o que mostra que o uso de tais algoritmos é bastante relevante em computação. Entretanto, os algoritmos probabilísticos podem retornar uma resposta errada com uma certa probabilidade. Observe, ainda que o uso de algoritmos probabilísticos não resolve problemas não computáveis.

A Complexidade Computacional caracteriza a complexidade de um problema a partir da quantidade de recursos computacionais, como espaço e tempo, para resolvê-lo. Problemas que tem a mesma complexidade compõem uma classe. As classes de complexidade computacional são relacionadas através de uma hierarquia.

A Complexidade Descritiva usa lógicas para expressar os problemas e capturar classes de complexidade computacional no sentido de expressar todos, e apenas, os problemas desta classe. Dessa forma, a complexidade de um problema não depende de fatores físicos, como tempo e espaço, mas apenas da expressividade da lógica que o define. Resultados importantes da área mostraram que várias classes de complexidade computacional podem ser caracterizadas por lógicas. Por exemplo, a classe NP foi mostrada equivalente à classe dos problemas expressos pelo fragmento existencial da Lógica de Segunda Ordem. Este estreito relacionamento entre tais áreas permite que alguns resultados da área de Lógica sejam transferidos para a de Complexidade Computacional e vice-versa.

Apesar da importância de algoritmos probabilísticos e da Complexidade Descritiva, existem poucos resultados de caracterização, por lógicas, das classes de complexidade computacional probabilísticas.

Neste trabalho, buscamos mostrar caracterizações para cada uma das classes de complexidade probabilísticas de tempo polinomial. Nos nossos resultados, utilizamos quantificadores generalizados de segunda ordem para simular a aceitação das máquinas não-determinísticas dessas classes. Achamos caracterizações lógicas na literatura apenas para as classes PP e BPP. No primeiro caso, a lógica utilizada era a de primeira ordem adicionada de um quantificador maioria de segunda ordem. Com a abordagem criada neste trabalho, conseguimos obter uma prova alternativa para a caracterização de PP. Com essa mesma metodologia, também conseguimos caracterizar a classe $\oplus P$ através de uma lógica com um quantificador de paridade. No caso de BPP, existia um resultado que utilizava uma lógica com semântica probabilística. Usando nossa abordagem de quantificadores generalizados, conseguimos obter uma caracterização alternativa para essa classe. Com o mesmo método, conseguimos caracterizar as classes probabilísticas semânticas RP, $coRP$, ZPP e a classe semântica $NP \cap coNP$. Por fim, mostramos uma aplicação dos resultados de Complexidade Descritiva na criação de algoritmos através de uma especificação lógica.

Palavras-chave: Complexidade Descritiva. Classes de Complexidade Probabilísticas. Quantificadores Generalizados.

ABSTRACT

Many computable problems can be solved more efficiently or in a more natural way through probabilistic algorithms, which shows that the use of such algorithms is quite relevant in Computer Science. However, probabilistic algorithms may return a wrong answer with a certain probability. Also, the use of probabilistic algorithms does not solve problems that are not computable.

In Computational Complexity, the complexity of a problem is characterized based on the amount of computational resources, such as space and time, needed to solve it. Problems that have the same complexity compose the same class. The computational complexity classes are related by a hierarchy.

In Descriptive Complexity, a logic is used to express problems and capture computational complexity classes in order to express all and only the problems of this class. Thus, the complexity of a problem does not depend on physical factors, such as time and space, but only on the expressiveness of the logic that defines it. Important results of the area states that several classes of computational complexity can be characterized by a logic. For example, the class NP has been shown equivalent to the class of problems expressed by the existential fragment of Second-Order Logic. This close relationship between these areas allows some results about Logics to be transferred to Computational Complexity and vice versa.

Despite of the importance of probabilistic algorithms and of Descriptive Complexity, there are few results on the characterization, by a logic, of probabilistic computational complexity classes.

In this work, we show characterizations for each of the polinomial time probabilistic complexity classes. In our results, we use second-order generalized quantifiers to simulate the acceptance of the nondeterministic machines of these classes. We found Logical characterizations in the literature only for classes PP and BPP. In the first case, the logic employed was the first-order added by a quantifier most of second-order. With the approach established in this work, we obtain an alternative proof for the characterization of PP. With the same methodology, we also characterize the class $\oplus P$ through a logic with a second-order parity quantifier. In the case of BPP , there was a result that used a logic with probabilistic semantics. Using our approach of generalized quantifiers, we obtain an alternative characterization for this class. With the same method, we were able to characterize the probabilistic semantic classes RP, $coRP$, ZPP and the semantic class $NP \cap coNP$. Finally, we show an application of Descriptive Complexity results in the creation of algorithms from a logic specification.

Keywords: Descriptive Complexity. Probabilistic Complexity Classes. Generalized Quantifiers.

LISTA DE FIGURAS

Figura 3.1 Classes de Complexidade 39

LISTA DE DEFINIÇÕES, PROPOSIÇÕES, LEMAS, TEOREMAS, COROLÁRIOS E EXEMPLOS

2.1.1 Definição (Alfabeto) 17
2.1.2 Definição (Vocabulário) 17
2.1.3 Definição (Termos) 18
2.1.4 Definição (Fórmulas) 18
2.1.5 Definição (Estrutura) 18
2.1.6 Definição (Atribuição) 18
2.1.7 Definição (Interpretação) 19
2.1.8 Definição 19
2.1.9 Definição (Interpretação dos Termos) 19
2.1.10 Definição (Relação de Satisfação) 19
2.1.11 Definição (Expansão de Estruturas) 20
2.1.12 Definição (Isomorfismo entre Estruturas) 20
2.2.1 Definição (Lógica de Segunda Ordem) 21
2.2.2 Definição (Atribuição de Segunda Ordem) 21
2.2.3 Definição (Relação de Satisfação) 21
2.3.1 Definição (Quantificador de Lindström) 22
2.3.1 Exemplo 23
2.3.2 Definição (Estrutura de Segunda Ordem) 23
2.3.3 Definição (Isomorfismo entre Estruturas de Segunda Ordem) 24
2.3.4 Definição (Quantificador de Segunda Ordem) 24
2.3.2 Exemplo 24
2.3.5 Definição 25
2.4.1 Definição (Operador) 25
2.4.2 Definição (Ponto Fixo) 25
2.4.3 Definição (Menor Ponto Fixo) 26
2.4.4 Definição (Operador Monótono) 26
2.4.1 Teorema 26

2.4.5 Definição (Lógica $FO(LFP)$) 26

3.1.1 Definição (Máquina de Turing Determinística) 28
3.1.2 Definição 28
3.1.3 Definição (Computação da Máquina de Turing) 29
3.1.4 Definição (M decide L) 29
3.1.5 Definição (M reconhece L) 29
3.1.6 Definição (Classe P) 30
3.1.7 Definição (Redução) 30
3.1.8 Definição (Problema Completo para uma Classe) 30
3.1.9 Definição (Máquina de Turing Não-Determinística) 30
3.1.10 Definição (N decide L) 31
3.1.11 Definição (Classe NP) 31
3.1.12 Definição (Classe $coNP$) 31
3.1.13 Definição (Classe $NP \cap coNP$) 32
3.2.1 Definição (N é precisa) 32
3.2.2 Definição (Máquina de Turing polinomial de Monte Carlo) 32
3.2.3 Definição (Classe RP) 33
3.2.4 Definição (Classe RP_ε) 33
3.2.1 Teorema 34
3.2.2 Teorema ($RP \subseteq NP$) 35
3.2.3 Teorema ($P \subseteq RP$) 35
3.2.4 Teorema 36
3.2.5 Definição (Classe $coRP$) 36
3.2.6 Definição (Classe ZPP) 37
3.2.7 Definição (Classe BPP) 37
3.2.5 Teorema ($RP \subseteq BPP$.) 37
3.2.8 Definição (Classe BPP_ε) 37
3.2.6 Teorema 37
3.2.9 Definição (Classe PP) 38
3.2.7 Teorema 38

3.2.8 Teorema ($NP \subseteq PP$) 39

4.1.1 Definição (Consultas) 42
4.1.2 Definição (Consultas Booleanas) 42
4.1.1 Exemplo 43
4.1.2 Exemplo 43
4.1.3 Definição (Consultas de Primeira Ordem) 43
4.1.4 Definição (Codificação) 44
4.1.5 Definição (Máquina de Turing M computa consulta Q) 44
4.1.6 Definição ($Q_b \in \mathscr{C}$) 44
4.1.7 Definição ($Q \in \mathscr{C}$) 45
4.1.8 Definição ($\mathscr{L}$ está em $\mathscr{C}$) 45
4.1.9 Definição ($\mathscr{L}$ captura $\mathscr{C}$) 45
4.1.1 Teorema ($FO \subseteq L$) 45
4.1.2 Teorema ($SO\exists = NP$) 46
4.1.3 Exemplo 47
4.1.4 Exemplo 48
4.1.3 Teorema ($FO(LFP) = P$) 48
4.2.1 Definição (Problema de Contagem) 49
4.2.2 Definição (Classe $\#P$) 49
4.2.3 Definição (Classe $\#FO$) 49
4.2.1 Exemplo 50
4.2.2 Exemplo 50
4.2.1 Teorema ($\#FO = \#P$) 50

5.1.1 Proposição 52
5.1.2 Proposição 52
5.1.3 Proposição 52
5.1.1 Teorema 53
5.2.1 Definição 54
5.2.1 Teorema 54

5.3.1 Definição (Lógica $FO(Most)_{qr\leq 1}$) 55
5.3.1 Teorema ($FO(Most)_{qr\leq 1} = PP$) 55
5.4.1 Definição (Classe $\oplus P$) 57
5.4.2 Definição (Lógica $FO(\oplus)_{qr\leq 1}$) 57
5.4.3 Definição 58
5.4.4 Definição 58
5.4.1 Exemplo 58
5.4.1 Teorema ($FO(\oplus)_{qr\leq 1} = \oplus P$) 58

6.1.1 Definição (Classes de Expansões de um Estrutura) 60
6.1.2 Definição (Lógica $P_{(\alpha,\beta]}L$) 61
6.1.3 Definição (Lógica $IFP(\mathscr{R})$) 61
6.1.1 Teorema ($BPIFP(\mathscr{R}) = BPP$) 61
6.2.1 Definição (Lógica $BPFO(Q_{\geq \frac{3}{4}})$) 62
6.2.2 Definição 62
6.2.1 Teorema ($BPFO(Q_{\geq \frac{3}{4}}) = BPP$) 62
6.2.1 Corolário 63
6.3.1 Proposição 63
6.3.2 Proposição 63
6.3.3 Proposição 64
6.3.4 Proposição 64
6.4.1 Definição (Lógica $RFO(Most_{=})$) 64
6.4.2 Definição 64
6.4.1 Teorema ($RFO(Most_{=}) = RP$) 65
6.4.1 Corolário 65
6.6.1 Definição (Lógica $coRFO(Most_{=})$) 67
6.6.2 Definição 67
6.6.1 Teorema ($coRFO(Most_{=}) = coRP$) 67
6.7.1 Definição (Lógica $ZPFO(Most_{=})$) 68
6.7.2 Definição 68
6.7.1 Teorema ($ZPFO(Most_{=}) = ZPP$) 68

6.8.1 Definição (Lógica $(N \cap coN)FO(\exists, \forall)$) 69
6.8.2 Definição 70
6.8.1 Teorema ($(N \cap coN)FO(\exists, \forall) = NP \cap coNP$) 70

SUMÁRIO

1 Introdução **14**

2 Lógica e Quantificadores Generalizados **17**

2.1 Lógica de Primeira Ordem 17

2.2 Lógica de Segunda Ordem 20

2.3 Quantificadores Generalizados 21

2.4 Lógica com Operador de Menor Ponto Fixo 25

3 Complexidade Computacional **27**

3.1 Máquinas de Turing e Classes de Complexidade 27

3.2 Máquinas de Turing Probabilísticas e Classes de Complexidade Probabilísticas . . 32

4 Complexidade Descritiva **41**

4.1 Noções Preliminares e Caracterização das Classes P e NP 41

4.2 Caracterização da Classe de Contagem $\#P$ 49

5 Complexidade Descritiva das Classes Sintáticas PP e $\oplus P$ **52**

5.1 $FO(Most^k)$ e Definibilidade para o Quantificador $Most^k$ 52

5.2 Caracterização da Classe PP 54

5.3 Uma Prova Alternativa de Caracterização da Classe PP e o $MAJSAT$ 55

5.4 Caracterização da Classe $\oplus P$ Através do Quantificador $\oplus$ 57

6 Complexidade Descritiva de Classes Probabilísticas Semânticas de Tempo Polinomial e da Classe $NP \cap coNP$ **60**

6.1 Caracterização da Classe BPP 60

6.2 Caracterização de BPP com Quantificadores Generalizados 62

6.3 Definibilidade com o Quantificador $Most^k_{=}$ 63

6.4 Caracterização da Classe RP 64

6.5 Expressando Problemas em RP com $RFO(Most_{=})$ 65

6.6 Caracterização da Classe Probabilística Semântica $coRP$ 67

6.7 Caracterização da Classe Probabilística Semântica ZPP 68
6.8 Caracterização da Classe Semântica $NP \cap coNP$ 69

7 Conclusões e Trabalhos Futuros **71**

Referências Bibliográficas **74**

Apêndice A – Criação Automática de Algoritmos de Tempo Polinomial Usando Complexidade Descritiva **77**

1 INTRODUÇÃO

A necessidade da probabilidade na Computação surge na existência de problemas computacionais que são resolvidos de forma mais natural ou mais eficiente com algoritmos que utilizam probabilidade. Em contrapartida de uma maior eficiência, alguns algoritmos probabilísticos podem retornar uma resposta errada. A probabilidade do erro pode ser diminuída executando o algoritmo várias vezes. Como exemplo de algoritmos probabilísticos, podemos citar o algoritmo Quicksort Randômico para ordenar uma lista de elementos, o algoritmo probabilístico para o Problema da Contratação vistos em (CORMEN et al., 2009) e o algoritmo probabilístico para o Problema do Corte Mínimo (MOTWANI; RAGHAVAN, 1995; MITZENMACHER; UPFAL, 2005). Diversas aplicações da computação utilizam algoritmos probabilísticos. Por exemplo, vários algoritmos de Inteligência Artificial E Big Data utilizam alguma abordagem probabilística (RUSSELL; NORVIG, 2010; RAJARAMAN; ULLMAN, 2011).

Para estudar a eficiência computacional de problemas computacionais quaisquer existe a área da Complexidade Computacional (PAPADIMITRIOU, 1994). Nessa área os problemas são classificados de acordo com a eficiência do melhor algoritmo que os resolvem. Por exemplo, um problema P_1 é mais complexo que outro problema P_2 se o melhor algoritmo que resolve P_1 necessita assintoticamente de mais recursos computacionais que o melhor algoritmo para P_2. A Complexidade Computacional também agrupa os problemas que tem dificuldade semelhante em classes de complexidade. Problemas com algoritmos probabilísticos também são classificados nessas classes.

Algumas questões da Complexidade Computacional estão abertas há bastante tempo. Por exemplo, podemos citar o famoso problema de saber se $P = NP$. No caso probabilístico temos o problema de saber se $P = BPP$, onde BPP é a classe de problemas que são resolvidos por máquinas de Turing probabilísticas em tempo polinomial com erro limitado.

Alguns problemas em aberto da Complexidade Computacional, como o problema de saber se $NPSPACE = coNPSPACE$, foram respondidos depois da criação de uma área chamada Complexidade Descritiva. O resultado acima e outros foram provados de forma independente por (IMMERMAN, 1988; SZELEPCSÉNYI, 1988). A Complexidade Descritiva pretende caracterizar a complexidade de um problema com base na linguagem lógica necessária para expressá-lo em vez de medidas físicas como tempo e espaço, e relacionar a linguagem utilizada com as classes de complexidade computacional. A semântica de uma lógica é dada por algoritmos e, dessa forma, a Complexidade Descritiva estaria relacionando esses algoritmos com as classes de complexidade computacional. Podemos dizer que essa área é uma "ponte" entre o estudo de Lógicas e de Teoria da Complexidade. Usando essa "ponte", alguns resultados de uma área podem ser traduzidos como resultados da outra.

O resultado que iniciou a área foi o que diz que os problemas que podem ser resolvidos por uma máquina de Turing não-determinística em tempo polinomial, ou seja, problemas da classe NP, são os mesmos que podem ser expressos pela lógica existencial de segunda ordem. Este é o famoso teorema de Fagin (FAGIN, 1973).

Uma aplicação de resultados desse tipo é saber a eficiência de problemas que envol-

vem lógica como, por exemplo, checar se uma fórmula é satisfeita por uma estrutura de entrada. Outra importância da Complexidade Descritiva seria a abordagem alternativa para provar resultados da Complexidade Computacional.

Mesmo com essa importância da Complexidade Descritiva e da probabilidade para a Ciência da Computação, poucos resultados foram obtidos sobre a Complexidade Descritiva de classes probabilísticas. Em (KONTINEN, 2009) foi mostrada uma caracterização para a hierarquia de contagem CH através da Lógica de Primeira Ordem acrescida do quantificador maioria $Most$ de segunda ordem. Dentro da hierarquia de contagem está a classe probabilísticas PP e como corolário essa classe é caracterizada pela Lógica formada pelas fórmulas sem aninhamento do quantificador $Most$.

Uma forma de definir lógicas probabilísticas de maneira similar às classes probabilísticas foi feita em (EICKMEYER; GROHE, 2010). Nesse mesmo trabalho, foi definida a lógica probabilística com ponto fixo inflacionário e contagem $BPIFP+C$ que captura a classe BPP mesmo em estruturas não ordenadas. O autor afirma que, de forma semelhante, é possível definir a lógica $BPDTC+C$ para caracterizar a classe probabilística de espaço logarítmico BPL. Na tese de doutorado do mesmo autor (EICKMEYER, 2011), além dos resultados mencionados acima, foi definida a lógica probabilística $BPFO$ que captura classe probabilística $BPAC^0$ em estruturas ordenadas.

O presente trabalho apresenta uma abordagem para a caracterização de classes probabilísticas de tempo polinomial através de lógicas com quantificadores generalizados de segunda ordem. Essa abordagem pode ser vista como uma generalização das abordagens dos trabalhos mencionados anteriormente (KONTINEN, 2009; EICKMEYER; GROHE, 2010) e da caracterização de $\#P$ apresentada em (THAKUR, 1992; SALUJA; SUBRAHMANYAM; THAKUR, 1995). Mostramos uma prova alternativa da caracterização de PP para mostrar nossa abordagem e em seguida expressamos um problema completo dessa classe na lógica utilizada. Em seguida, usamos nossa abordagem para caracterizar a classe de problemas com um número ímpar de certificados $\oplus P$ que não é probabilística. Nossa caracterização usa a Lógica de Primeira Ordem com o quantificador generalizado ímpar $\oplus$ e também expressamos um problema completo da classe com essa lógica.

Também utilizamos nossa abordagem para obter uma caracterização alternativa da classe BPP e caracterizações das classes RP, $coRP$ e ZPP. Resultados de Complexidade Descritiva para essas classes não foram encontrados na literatura. Problemas da classe RP foram expressos com a lógica utilizada na caracterização. Também identificamos algumas dificuldades e suas causas na definibilidade de problemas dessa classe. Algumas soluções possíveis são apresentadas mas deixamos sua execução como trabalho futuro.

Para mostrar a flexibilidade da abordagem, caracterizamos também a classe não probabilística $NP \cap coNP$. Além desses resultados de caracterização de classes, vamos mostrar, no apêndice, como podemos usar os resultados da Complexidade Descritiva para a criação automática de algoritmos dada uma especificação lógica. Como a semântica de uma lógica é definida por um algoritmo, podemos entender esse resultado como uma explicitação do algoritmo da semântica. Por último, vamos motivar a investigação de novos resultados relacionando

lógicas e outras classes de complexidade probabilísticas.

O trabalho está organizado da seguinte maneira: No segundo capítulo, vamos apresentar uma introdução de Lógica e de Quantificadores Generalizados. No terceiro capítulo, vamos apresentar uma introdução à Complexidade Computacional mostrando também classes de complexidade probabilísticas. No capítulo quatro, mostramos uma visão geral da área de Complexidade Descritiva. Em seguida, nos capítulos 5 e 6, vamos definir algumas lógicas com quantificadores de segunda ordem generalizados para caracterizar as classes mencionadas acima. No capítulo 5 vamos mostrar os resultados para as classes sintáticas e no capítulo 6 os das classes semânticas. Por último, vamos apresentar as conclusões do trabalho e definir possíveis trabalhos futuros.

2 LÓGICA E QUANTIFICADORES GENERALIZADOS

Neste capítulo nós vamos fazer uma breve introdução sobre Lógica. Na primeira seção vamos apresentar a Lógica Clássica de Primeira Ordem (EBBINGHAUS; FLUM; THOMAS, 1994), aqui simplesmente denominada Lógica de Primeira Ordem, e nas seções seguintes vamos mostrar algumas linguagens lógicas que podem estender a linguagem da Lógica de Primeira Ordem.

2.1 Lógica de Primeira Ordem

Para definir a linguagem da Lógica de Primeira Ordem é necessário primeiro definirmos o alfabeto dessa linguagem.

Definição 2.1.1 (Alfabeto)**.** *O alfabeto da linguagem da Lógica de Primeira Ordem possui os seguintes símbolos:*

- *Um conjunto enumerável de variáveis VAR:* $x, y, z, x_1, y_1, z_1, x_2, ...$
- *Um conjunto de símbolos lógicos:* $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists$
- *O símbolo de igualdade:* $\equiv$
- *Os símbolos de pontuação:* $(,)$
- *Um conjunto* τ *de símbolos de constantes e símbolos de relação e função de qualquer aridade natural.*

Para os símbolos de constantes vamos utilizar letras minúsculas com ou sem subsestrito, por exemplo, $a, b, c, a_1, b_1, c_1 ....$ Nos símbolos funcionais vamos utilizar também letras minúsculas com ou sem subescrito mas com a aridade superescrita: $f^{a_1}, g^{a_2}, h^{a_3}, f_1^{a_4}, g_1^{a_5}, h_1^{a_6} ....$ Vamos utilizar as letras maiúsculas com aridade superescrita e com ou sem índice subescrito para representar os símbolos relacionais: $R^{b_1}, S^{b_2}, P^{b_3}, R_1^{b_4}, S_1^{b_5}, P_1^{b_6}$. Quando não houver dúvida, vamos omitir as aridades dos símbolos de funções e relações.

Definição 2.1.2 (Vocabulário)**.** *Os únicos símbolos que mudam no alfabeto são os símbolos de* τ*. Dependendo dos símbolos de* τ *podemos obter fórmulas diferentes. Chamamos o conjunto* τ *de vocabulário.*

Podemos representar um vocabulário da forma a seguir:

$$\tau = \left\langle R_1^{a_1}, R_2^{a_2}, ..., f_1^{b_1}, f_2^{b_2}, ..., c_1, c_2, ... \right\rangle,$$

onde cada a_i e cada b_i é a aridade do símbolo relacional correspondente. Pelo que foi mostrado na representação acima o vocabulário pode ser infinito. A seguir vamos definir os termos. Os termos são os elementos que podemos falar sobre na linguagem. Veja que os termos dependem do vocabulário τ.

Definição 2.1.3 (Termos)**.** *Seja τ um vocabulário. Os termos de τ são obtidos através de um número finito de aplicações das regras a seguir:*

- *Toda variável é um τ-termo.*
- *Toda constante em τ é um τ-termo.*
- *Se t_1, ..., t_k são τ-termos e f é um símbolo de função k-ária em τ então $f(t_1,...,t_k)$ é um τ-termo.*

Agora que definimos os termos, podemos definir propriedades sobre os termos. A seguir vamos definir essas propriedades sobre os termos como fórmulas:

Definição 2.1.4 (Fórmulas)**.** *Seja τ um vocabulário. As τ-fórmulas são precisamente obtidas por um número finito de aplicações das seguintes regras:*

- *Se t_1 e t_2 são τ-termos então $t_1 \equiv t_2$ é uma τ-fórmula.*
- *Se t_1, ..., t_k são τ-termos e R é um símbolo relacional de aridade k de τ então $R(t_1,...,t_k)$ é uma τ-fórmula.*
- *Se φ é uma τ-fórmula então $\neg\varphi$ é uma τ-fórmula.*
- *Se φ e ψ são τ-fórmulas então $(\varphi \circ \psi)$ é uma τ-fórmula, onde $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.*
- *Se φ é uma τ-fórmula e x é uma variável então $\forall x\varphi$ e $\exists x\varphi$ são τ-fórmulas.*

Para verificar se uma fórmula é verdadeira ou falsa precisamos determinar os elementos que vão ser considerados, as relações, funções e constantes que vão representar cada um dos símbolos relacionais, funcionais e de constantes. Além disso, precisamos atribuir elementos do domínio para as variáveis.

Uma estrutura vai determinar os elementos do domínio e as relações, constantes e funções. Vamos definir a seguir as τ-estruturas e em seguida as atribuições.

Definição 2.1.5 (Estrutura)**.** *Seja τ um vocabulário. Uma τ-estrutura $\mathscr{A}$ é uma tupla*

$$\mathscr{A} = \langle A, R_1^{\mathscr{A}}, ..., f_1^{\mathscr{A}}, c_1^{\mathscr{A}}, ... \rangle,$$

onde A é o domínio, cada $R_i^{\mathscr{A}}$ é uma relação sobre A com aridade definida pelo símbolo relacional R_i, cada $f_i^{\mathscr{A}}$ é uma função sobre A com aridade definida pelo símbolo funcional correspondente do vocabulário τ e cada $c_i^{\mathscr{A}}$ é um elemento do domínio A.

Definição 2.1.6 (Atribuição)**.** *Seja $\mathscr{A}$ uma estrutura. Uma atribuição é uma mapeamento de variáveis em elementos do domínio A de $\mathscr{A}$. Ou seja, $\beta : VAR \rightarrow A$ é uma atribuição.*

Abaixo vamos definir as interpretações.

Definição 2.1.7 (Interpretação)**.** *Uma τ-interpretação é um par $(\mathscr{A},\beta)$ consistindo de uma τ-estrutura $\mathscr{A}$ e uma atribuição β em $\mathscr{A}$.*

Quando um conjunto de símbolos τ é claro pelo contexto vamos utilizar apenas os termos estruturas e interpretações no lugar de τ-estruturas e τ-interpretações. As duas próximas definições vão ser úteis para definirmos a noção de satisfação.

Definição 2.1.8. *Se β é uma atribuição em $\mathscr{A}$, $a \in A$ e x é uma variável então definimos como $\beta a/x$ a atribuição que mapeia x para a e se comporta como β para qualquer outra variável diferente de x.*

Se $\mathfrak{I} = (\mathscr{A},\beta)$ então definimos $\mathfrak{I}a/x = (\mathscr{A},\beta a/x)$.

Definição 2.1.9 (Interpretação dos Termos)**.** *Seja τ um vocabulário e $\mathfrak{I}$ uma interpretação formada pela τ-estrutura $\mathscr{A}$ e pela atribuição β. A interpretação dos termos é feita indutivamente a seguir:*

Para uma variável x seja $\mathfrak{I}(x) = \beta(x)$

Para uma constante $c \in \tau$ seja $\mathfrak{I}(c) = c^{\mathscr{A}}$

Para um símbolo de função k-ário $f \in \tau$ e termos $t_1,...,t_k$ seja $\mathfrak{I}(f(t_1,...,t_k)) = f^{\mathscr{A}}(\mathfrak{I}(t_1),...,\mathfrak{I}(t_k))$.

Agora podemos definir a relação de satisfação. Essa definição nos permite saber quando uma interpretação satisfaz uma fórmula, ou seja, quando uma fórmula é verdadeira de acordo com uma interpretação.

Definição 2.1.10 (Relação de Satisfação)**.** *Para uma interpretação $\mathfrak{I} = (\mathscr{A},\beta)$ definimos a relação de satisfação indutivamente nas fórmula da forma abaixo:*
$\mathfrak{I} \models t_1 \equiv t_2$ *se e somente se* $\mathfrak{I}(t_1) = \mathfrak{I}(t_2)$
$\mathfrak{I} \models R(t_1,...,t_k)$ *se e somente se* $\mathfrak{I}(t_1),...,\mathfrak{I}(t_k) \in R^{\mathscr{A}}$
$\mathfrak{I} \models \neg\varphi$ *se e somente se* $\mathfrak{I} \not\models \varphi$
$\mathfrak{I} \models (\varphi \wedge \psi)$ *se e somente se* $\mathfrak{I} \models \varphi$ *e* $\mathfrak{I} \models \psi$
$\mathfrak{I} \models (\varphi \vee \psi)$ *se e somente se* $\mathfrak{I} \models \varphi$ *ou* $\mathfrak{I} \models \psi$
$\mathfrak{I} \models (\varphi \rightarrow \psi)$ *se e somente se (se* $\mathfrak{I} \models \varphi$ *então* $\mathfrak{I} \models \psi$*)*
$\mathfrak{I} \models (\varphi \leftrightarrow \psi)$ *se e somente se (*$\mathfrak{I} \models \varphi$ *se e somente se* $\mathfrak{I} \models \psi$*)*
$\mathfrak{I} \models \exists x\varphi$ *se e somente se existe um* $a \in A$ *tal que* $\mathfrak{I}a/x \models \varphi$
$\mathfrak{I} \models \forall x\varphi$ *se e somente se para todo* $a \in A$ *temos* $\mathfrak{I}a/x \models \varphi$.

Note que a validade de uma fórmula φ sobre uma interpretação $\mathfrak{I}$ depende da atribuição apenas para a quantidade finita de variáveis livres ocorrendo em φ. Se as variáveis ocorrendo livres são $x_1,x_2,...,x_n$ então os valores da atribuição significantes são os $a_i = \beta(x_i)$ para $i \in \{1,2,...,n\}$. Nesse caso, podemos usar a notação abaixo:

$$\mathscr{A} \models \varphi(a_1,...,a_n).$$

E no caso de sentenças, ou seja, fórmulas sem variável livre, podemos fazer apenas

$$\mathscr{A} \models \varphi.$$

A definição a seguir vai ser utilizada quando apresentarmos o resultado existente de (EICKMEYER; GROHE, 2010) para definir as lógicas probabilísticas.

Definição 2.1.11 (Expansão de Estruturas)**.** *Seja $\mathscr{A}$ uma estrutura de vocabulário τ e seja τ' um vocabulário tal que $\tau \subset \tau'$. Uma estrutura $\mathscr{A}'$ é uma expansão de $\mathscr{A}$ quando $A = A'$ e a interpretação dos elementos das duas estruturas concordam no vocabulário τ.*

Agora vamos definir isomorfismo entre estruturas e classe de estruturas fechadas sobre isomorfismo. Vamos utilizar essas definições para apresentar os quantificadores generalizados da seção 2.3.

Definição 2.1.12 (Isomorfismo entre Estruturas)**.** *Seja $\mathscr{A}$ e $\mathscr{B}$ duas estruturas de vocabulário τ. Um mapeamento $\pi : A \to B$ é chamado de isomorfismo de $\mathscr{A}$ em $\mathscr{B}$ se*

π é uma bijeção.

Para qualquer relação $R \in \tau$ de aridade n e $a_1, ..., a_n \in A$

$$R^{\mathscr{A}}(a_1, ..., a_n) \text{ se e somente se } R^{\mathscr{B}}(\pi(a_1), ..., \pi(a_n))$$

Para qualquer função $R \in \tau$ de aridade n e $a_1, ..., a_n \in A$

$$\pi(f^{\mathscr{A}}(a_1, ..., a_n)) = f^{\mathscr{B}}(\pi(a_1), ..., \pi(a_n))$$

Para qualquer constante $c \in \tau$

$$\pi(c^{\mathscr{A}}) = c^{\mathscr{B}}$$

2.2 Lógica de Segunda Ordem

A diferença básica da Lógica Clássica de Segunda Ordem, ou simplesmente Lógica de Segunda Ordem, com relação à Lógica de Primeira Ordem é a capacidade de quantificar sobre relações como, por exemplo, subconjuntos do domínio. A seguir, vamos definir a sintaxe e semântica da Lógica de Segunda Ordem. As definições mostradas nesta seção podem ser encontradas em (EBBINGHAUS; FLUM; THOMAS, 1994).

No alfabeto da Lógica de Segunda Ordem temos, além do alfabeto convencional da Lógica de Primeira Ordem, variáveis de segunda ordem $X, Y, Z, ...$ de qualquer aridade k natural.

A seguir vamos definir indutivamente as fórmulas da Lógica de Segunda Ordem estendendo a definição das fórmulas da Lógica de Primeira Ordem.

Definição 2.2.1 (Lógica de Segunda Ordem)**.** *Na Lógica de Segunda Ordem as regras de formação de fórmulas da Lógica de Primeira Ordem são adicionadas das duas regras a seguir:*

- *Se X é uma variável relacional de aridade k e $t_1,...,t_k$ são τ-termos então $X(t_1,...,t_k)$ é uma τ-fórmula.*
- *Se φ é uma τ-fórmula e X uma variável relacional então $\exists X\varphi$ e $\forall X\varphi$ são τ-fórmulas.*

Agora vamos definir uma atribuição para as variáveis relacionais que adicionamos ao alfabeto.

Definição 2.2.2 (Atribuição de Segunda Ordem)**.** *Uma atribuição de segunda ordem γ é um mapeamento de variáveis de primeira ordem v_i e variáveis de segunda ordem V_i^k em elementos do domínio A de uma estrutura $\mathscr{A}$ e em relações k-árias em A, respectivamente.*

Podemos estender a relação de satisfação da Lógica de Primeira Ordem definindo a satisfação das fórmulas adicionadas pelas regras de formação de fórmulas acima da seguinte forma:

Definição 2.2.3 (Relação de Satisfação)**.** *Seja $\mathscr{A}$ uma τ-estrutura, γ uma atribuição de segunda ordem, $\mathfrak{I} = (\mathscr{A}, \gamma)$ uma interpretação e X uma variável relacional de aridade k então*

- $\mathfrak{I} \models X(t_1,...,t_k)$ *se e somente se* $(t_1^{\mathscr{A}},...,t_k^{\mathscr{A}}) \in \gamma(X)$.
- $\mathfrak{I} \models \exists X\varphi$ *se e somente se existe um* $R \subseteq A^k$ *tal que* $\mathfrak{I}R/X \models \varphi$.
- $\mathfrak{I} \models \forall X\varphi$ *se e somente se para todo* $R \subseteq A^k$ *temos* $\mathfrak{I}R/X \models \varphi$.

Onde $\mathfrak{I}R/X = (\mathscr{A}, \gamma R/X)$ e $\gamma R/X$ é uma atribuição que mapeia X para R mas para outras variáveis de segunda ordem concorda com γ.

2.3 Quantificadores Generalizados

Os Quantificadores Generalizados foram introduzidos na lógica como generalizações dos quantificadores padrão $\exists$ e $\forall$. O primeiro trabalho a considerar quantificadores diferentes dos usuais existencial e universal foi (MOSTOWSKI, 1957). Nesse trabalho, o autor apresentou uma forma de definir novos quantificadores com escopo de apenas uma variável e uma fórmula. Uma forma mais geral de definir quantificadores generalizados, representando quantificadores como classes de estruturas relacionais fechadas sob isomorfismo, foi introduzida em (LINDSTRÖM, 1966). Com essa abordagem, é possível definir quantificadores com escopo de várias variáveis e várias fórmulas.

Quantificadores generalizados são bastante estudados em trabalhos sobre linguagem natural como pode ser visto em (BARWISE; COOPER, 1981; PETERS; WESTERSTåHL, 2006; BADIA, 2009). Mais detalhes sobre quantificadores generalizados na Teoria dos Modelos Finitos podem ser encontrados em (EBBINGHAUS; FLUM, 1995).

Quantificadores Generalizados são fundamentais no nosso trabalho pois vamos utilizá-los para simular a aceitação das máquinas de Turing probabilísticas.

A Lógica de Primeira Ordem não é capaz de expressar, por exemplo, que uma fórmula vale para um número ímpar de elementos do domínio. Para estender a Lógica de Predicados com esse quantificador podemos adicionar um novo quantificador $\mathcal{Q}_{odd}$ com semântica dada a seguir:

$$\mathscr{A} \models \mathcal{Q}_{odd} x \varphi(x) \text{ se e somente se } |\{a \in A | \mathscr{A} \models \varphi(a)\}| \text{ é ímpar.}$$

Para definir a semântica de quantificadores generalizados de maneira geral, podemos utilizar classe de estruturas. Por exemplo, no caso do quantificador $\mathcal{Q}_{odd}$ podemos definir a classe de estruturas $\mathcal{Q}_{odd} = \{\langle A, P\rangle \,|P \subseteq A \text{ e } |P| \text{ é ímpar }\}$. Ou seja, $\mathcal{Q}_{odd}$ é uma classe de estruturas que tem um predicado com uma quantidade ímpar de elementos. Utilizamos $\mathcal{Q}_{odd}$ como símbolo de um quantificador da linguagem e também como um conjunto de estruturas. Utilizando essa classe de estruturas podemos definir a semântica do quantificador $\mathcal{Q}_{odd}$ da seguinte maneira:

$$\mathscr{A} \models \mathcal{Q}_{odd} x \varphi(x) \text{ se e somente se } \{a \in A | \mathscr{A} \models \varphi(a)\} \in \mathcal{Q}_{odd}.$$

De maneira geral, seja P um símbolo de predicado unário e K uma classe de estruturas com vocabulário $\{P\}$ que é fechada sob isomorfismo. A interpretação de um quantificador $\mathcal{Q}$ de Lindström que "varre" apenas uma variável e uma fórmula φ é dada por

$$\mathscr{A} \models \mathcal{Q} x \varphi(x) \text{ se e somente se } \langle A, \{a \in A | \mathscr{A} \models \varphi(a)\}\rangle \in K.$$

De fato, qualquer classe de estruturas com vocabulário τ que tenha apenas símbolos relacionais, ou seja, estruturas relacionais, permite a definição de um quantificador. As definições a seguir são as mesmas utilizadas em (KONTINEN, 2009).

Definição 2.3.1 (Quantificador de Lindström)**.** *Seja $s = \langle l_1, ..., l_r\rangle$ uma tupla de inteiros positivos. Um quantificador de Lindström de tipo s é uma classe $\mathcal{Q}$ de estruturas relacionais de vocabulário $\tau = \{P_1, ..., P_r\}$ tal que cada $P_i \in \tau$ tem aridade l_i e $\mathcal{Q}$ é fechado sob isomorfismo. A extensão $FO(\mathcal{Q})$ da Lógica de Primeira Ordem por um quantificador $\mathcal{Q}$ é definida como segue:*

- *As regras de formação de fórmulas são estendidas com a regra: Se para $1 \leq i \leq r$, $\varphi_i(\bar{x}_i)$ é uma fórmula e $\bar{x}_i$ é uma tupla de l_i variáveis distintas duas a duas então $\mathcal{Q}\bar{x}_1 ... \bar{x}_r(\varphi_1(\bar{x}_1), ..., \varphi_r(\bar{x}_r))$ é uma fórmula.*
- *A relação de satisfação da lógica é adicionada do seguinte caso:*

$$\mathscr{A} \models \mathcal{Q}\bar{x}_1 ... \bar{x}_r(\varphi_1(\bar{x}_1), ..., \varphi_r(\bar{x}_r)) \text{ se e somente se } \left\langle A, \varphi_1^{\mathscr{A}}, ..., \varphi_r^{\mathscr{A}}\right\rangle \in \mathcal{Q}, \text{ onde } \varphi_i^{\mathscr{A}} = \{\bar{a} \in A^{l_i} | \mathscr{A} \models \varphi_i(\bar{a})\}.$$

A seguir vamos ver alguns exemplos de quantificadores generalizados.

Exemplo 2.3.1. *O dois primeiros são os quantificadores universal e existencial usuais no formato de quantificadores de Lindström. O terceiro exemplo é o quantificador ímpar que descrevemos anteriormente. O quarto exemplo é o quantificador de Resher que "varre" duas variáveis e duas fórmulas e permite comparar a quantidade de elementos que satisfazem cada uma das duas fórmulas. Os dois últimos são exemplos de quantificadores usados na linguagem natural. Os dois querem expressar sentenças da forma "A maioria dos A's são B's" e "Algum A é B", respectivamente.*
$\forall = \{\langle A,P\rangle \,|P = A\}$
$\exists = \{\langle A,P\rangle \,|P \subseteq A \text{ e } P \neq \emptyset\}$
$\mathscr{Q}_{odd} = \{\langle A,P\rangle \,|P \subseteq A \text{ e } |P| \text{ é ímpar } \}$
$\mathscr{R} = \{\langle A,P,S\rangle \,|P,S \subseteq A \text{ e } |P| > |S|\}$
$\mathscr{Q}_{Most_r} = \{\langle A,P,S\rangle \,|P,S \subseteq A \text{ e } |P \cap S| > |P - S|\}$
$\mathscr{Q}_{Some} = \{\langle A,P,S\rangle \,|P,S \subseteq A \text{ e } |P \cap S| \neq \emptyset\}$.

Todos os quantificadores definidos anteriormente são quantificadores sobre variáveis de primeira ordem. Nosso próximo passo é definir quantificadores generalizados que quantificam sobre variáveis de segunda ordem. Alguns trabalhos que iniciaram o estudo de quantificadores generalizados de segunda ordem foram (ANDERSSON, 2002; KONTINEN, 2010, 2006).

(ANDERSSON, 2002) analisou a expressividade de quantificadores generalizados de segunda ordem em estruturas finitas. Ele mostrou que quase todas as lógicas contáveis com relações de aridade no máximo 2 são equivalentes à um fragmento de uma lógica $FO(\mathscr{Q})$, onde Q é algum quantificador generalizado de segunda ordem com escopo de apenas uma fórmula e uma variável relacional de aridade 1.

Em (KONTINEN, 2010, 2006), foram analisadas questões de definibilidade de quantificadores generalizados de segunda ordem. Por exemplo, saber se a classe de estruturas que define um quantificador $\mathscr{Q}$ é axiomatizável em uma lógica $\mathscr{L}$. Em (KONTINEN, 2009) foi feita uma caracterização lógica da hierarquia de contagem. A lógica definida em (KONTINEN, 2009) possui um quantificador generalizado de segunda ordem interpretado como a maioria das relações.

A seguir vamos definir quantificadores generalizados de segunda ordem. No nosso trabalho, vamos utilizar alguns quantificadores generalizados de segunda ordem para os resultados de caracterização das classes probabilísticas de tempo polinomial. Os quantificadores que vamos utilizar vão ser apresentados ainda nesta seção. As definições apresentadas nessa seção são as mesmas de (ANDERSSON, 2002; KONTINEN, 2009).

Definição 2.3.2 (Estrutura de Segunda Ordem)**.** *Seja $t = \langle s_1,...,s_m\rangle$ onde cada $s_i = \langle l^i_1, l^i_2, ..., l^i_{r_i}\rangle$ é uma tupla de naturais para $1 \leq i \leq m$. Uma estrutura de segunda ordem de tipo t é uma estrutura da forma $\langle A, P_1, ..., P_m\rangle$, onde para cada P_i temos $P_i \subseteq \mathscr{P}(A^{l^i_1}) \times ... \times \mathscr{P}(A^{l^i_{r_i}})$.*

Pela definição acima, cada P_i é uma tupla de conjuntos de conjuntos. Para exemplificar, vamos verificar um exemplo. Suponha um tipo $t = \langle s_1, s_2\rangle$ com $s_1 = \langle 2\rangle$ e $s_1 = \langle 1\rangle$. Uma

estrutura $\langle A, P_1, P_2 \rangle$ com domínio $A = \{1,2\}$, $P_1 = \{\emptyset, \{(1,1),(2,1)\}\}$ e $P_2 = \{\{1\},\{1,2\}\}$ é uma estrutura de segunda ordem de tipo t. Cada predicado é um conjunto de conjuntos. Cada elemento de um predicado é um conjunto que pode ser formado por tuplas de elementos do domínio A.

Para os propósitos do nosso trabalho, não vamos precisar de predicados de segunda ordem que sejam tuplas de conjuntos de conjuntos. Usando a definição acima vamos utilizar apenas que cada $P_i \subseteq \mathscr{P}(A^{l^i})$. Ou seja, para um tipo $t = \langle s_1, ..., s_m \rangle$, cada s_i é uma tupla de apenas um elemento. Nosso exemplo no parágrafo acima segue essa restrição.

Definição 2.3.3 (Isomorfismo entre Estruturas de Segunda Ordem)**.** *Uma classe $\mathscr{Q}$ de estruturas de segunda ordem de tipo t é fechada sob isomorfismo quando se $\langle A, P_1, ..., P_m \rangle \in \mathscr{Q}$ e $f : A \to B$ é uma bijeção tal que $S_i = \{(f[A_1], ..., f[A_{r_i}]) | (A_1, ..., A_{r_i}) \in P_i\}$ para $1 \leq i \leq m$ então $\langle B, S_1, ..., S_m \rangle \in \mathscr{Q}$. Onde $f[R] = \{(f(a_1), ..., f(a_k)) | (a_1, ..., a_k) \in R\}$ é a imagem da relação R sobre a função f.*

Definição 2.3.4 (Quantificador de Segunda Ordem)**.** *Um quantificador de segunda ordem $\mathscr{Q}$ de tipo t é uma classe de estruturas de segunda ordem de tipo t tal que $\mathscr{Q}$ é fechado sob isomorfismo.*

A seguir vamos mostrar exemplos de quantificadores generalizados de segunda ordem:

Exemplo 2.3.2. *O primeiro e o segundo quantificador são o universal e existencial k-ários de segunda ordem. O terceiro é o quantificador maioria que expressa que a maioria das relações k-árias de segunda ordem satisfazem a fórmula. O quarto quantificador é o maioria relativo e expressa que a quantidade de relações k-árias de segunda ordem na interseção é maior que a quantidade que está apenas na primeira relação. O quinto é a versão de segunda ordem do quantificador de Resher. O sexto é a versão de segunda ordem do quantificador $\mathscr{Q}_o dd$. Os próximos três quantificadores são semelhantes ao terceiro, quarto e quinto, respectivamente. A diferença é que na comparação temos um maior ou igual. Os dois últimos expressam que a quantidade de elementos é maior que $\frac{3}{4}$ do total e menor que $\frac{1}{4}$ do total, respectivamente. O valor 2^{A^k} representa a quantidade total de possibilidade para um predicado k-ário de segunda ordem.*

$\forall_2^k = \{\langle A, P \rangle | P = \mathscr{P}(A^k)\}$
$\exists_2^k = \{\langle A, P \rangle | P \subseteq \mathscr{P}(A^k) \text{ e } P \neq \emptyset\}$
$Most^k = \{\langle A, P \rangle | P \subseteq \mathscr{P}(A^k) \text{ e } |P| > 2^{A^k - 1}\}$
$Most_r^k = \{\langle A, P, S \rangle | P, S \subseteq \mathscr{P}(A^k) \text{ e } |P \cap S| > |P - S|\}$
$\mathscr{R}^k = \{\langle A, P, S \rangle | P, S \subseteq \mathscr{P}(A^k) \text{ e } |P| > |S|\}$
$\oplus^k = \{\langle A, P \rangle | P \subseteq \mathscr{P}(A^k) \text{ e } |P| \text{ é ímpar }\}$
$Most_=^k = \{\langle A, P \rangle | P \subseteq \mathscr{P}(A^k) \text{ e } |P| \geq 2^{A^k - 1}\}$
$Most_{r,=}^k = \{\langle A, P, S \rangle | P, S \subseteq \mathscr{P}(A^k) \text{ e } |P \cap S| \geq |P - S|\}$
$\mathscr{R}_=^k = \{\langle A, P, S \rangle | P, S \subseteq \mathscr{P}(A^k) \text{ e } |P| \geq |S|\}$
$Q_{\geq \frac{3}{4}}^k = \{\langle A, P \rangle | P \subseteq \mathscr{P}(A^k) \text{ e } |P| \geq 3 \times 2^{A^k - 2}\}$
$Q_{< \frac{1}{4}}^k = \{\langle A, P \rangle | P \subseteq \mathscr{P}(A^k) \text{ e } |P| < 2^{A^k - 2}\}$

A extensão $FO(\mathcal{Q})$ da Lógica de Primeira Ordem por um quantificador $\mathcal{Q}$ é definida a seguir:

Definição 2.3.5. *O conjunto de regras de formação de fórmulas da lógica $FO(\mathcal{Q})$ é adicionado da regra:*

- *Se $\varphi_i(\bar{X}_i)$ é uma fórmula para $1 \leq i \leq m$ e $\bar{X}_i = (X_{i_1}, ..., X_{i_{r_i}})$ é uma tupla de variáveis relacionais distintas duas a duas tal que a aridade de cada X_{i_j} é l^i_j para $1 \leq j \leq r_i$ então*

$$\mathcal{Q}\bar{X}_1 ... \bar{X}_m(\varphi_1(\bar{X}_1), ..., \varphi_m(\bar{X}_m)) \text{ é fórmula.}$$

A relação de satisfação da lógica é adicionada de mais um caso:

$$\mathcal{A} \models \mathcal{Q}\bar{X}_1 ... \bar{X}_m(\varphi_1(\bar{X}_1), ..., \varphi_m(\bar{X}_m)) \text{ se e somente se } \left\langle A, \varphi_1^{\mathcal{A}}, ..., \varphi_m^{\mathcal{A}} \right\rangle \in \mathcal{Q},$$

$$\text{onde } \varphi_i^{\mathcal{A}} = \{\bar{R} \in \mathcal{P}(A^{l^i_1}) \times ... \times \mathcal{P}(A^{l^i_{r_i}}) | \mathcal{A} \models \varphi_i(\bar{R})\}.$$

2.4 Lógica com Operador de Menor Ponto Fixo

Uma forma de expressar mais informações usando a Lógica de Primeira Ordem é adicionar a capacidade de definir novas relações por indução (IMMERMAN, 1999). Vamos apresentar a Lógica com Operador de Menor Ponto Fixo pois no capítulo 4 iremos mostrar que as sentenças dessa lógica correspondem exatamente aos problemas da classe P (GUREVICH; SHELAH, 1985). Além disso, no apêndice vamos mostrar uma aplicação do resultado mencionado acima criando um algoritmo que recebe uma sentença dessa lógica e retorna um algoritmo que resolve o problema da satisfação dessa fórmula de entrada.

Um exemplo importante de relação que pode ser definida indutivamente é o fecho transitivo. Seja $\tau_g = \left\langle E^2, s, t \right\rangle$ o vocabulário dos grafos, podemos definir o fecho reflexivo transitivo E^* de E da seguinte forma:

$$\varphi(R, x, y) := x \equiv y \vee \exists z(E(x, z) \wedge R(z, y)),$$

onde R é uma variável relacional. A fórmula acima pode ser vista como uma formalização da definição do fecho reflexivo transitivo de E. A ideia é que a relação R comece sendo interpretada como uma relação vazia e indutivamente incrementada de pares de elementos que satisfazem a fórmula apresentada acima. As definições e teoremas mostrados a seguir podem ser vistos em (IMMERMAN, 1999).

Definição 2.4.1 (Operador)**.** *Seja A um conjunto. Um operador é um mapeamento $F : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ na classe de todas as relações k-árias sobre A.*

Definição 2.4.2 (Ponto Fixo)**.** *Seja $F : \mathcal{P}(A^k) \to \mathcal{P}(A^k)$ um operador. Um ponto fixo de F é uma relação $R \subseteq A^k$ tal que $F(R) = R$.*

Em geral, um operador pode ter vários ou nenhum ponto fixo.

Definição 2.4.3 (Menor Ponto Fixo)**.** *Seja $F : \mathscr{P}(A^k) \to \mathscr{P}(A^k)$ um operador. Um menor ponto fixo de F é um ponto fixo R de F tal que $R \subseteq Y$ para cada ponto fixo Y de F.*

Definição 2.4.4 (Operador Monótono)**.** *Dizemos que um operador $F : \mathscr{P}(A^k) \to \mathscr{P}(A^k)$ é monótono quando para todo $X, Y \subseteq A^k$ se $X \subseteq Y$ então $F(X) \subseteq F(Y)$.*

Para qualquer estrutura $\mathscr{A}$, a interpretação $\varphi^{\mathscr{A}}(R,x,y)$ da fórmula $\varphi(R,x,y)$ induz um mapeamento de relações binárias para relações binárias no domínio de $\mathscr{A}$:

$$\varphi^{\mathscr{A}}(R,x,y) = \{\langle a,b\rangle \,|\mathscr{A} \models \varphi(R,a,b)\}$$

esse mapeamento é chamado monótono se para toda R, S,

$$\text{se } R \subseteq S \text{ então } \varphi^{\mathscr{A}}(R) \subseteq \varphi^{\mathscr{A}}(S).$$

Quando uma relação aparece no escopo de um número par de negações dizemos que a relação aparece positivamente na fórmula. Na fórmula $\varphi(R,x,y)$, R aparece positivamente. Disso, segue que o mapeamento $\varphi^{\mathscr{A}}(R)$ é monótono.

Seja $(\varphi^{\mathscr{A}}(R))^r$ significar $\varphi^{\mathscr{A}}(R)$ iterado r vezes.

Teorema 2.4.1. *Seja R um novo símbolo relacional de aridade k e seja $\varphi(R,x_1,...,x_k)$ monótona. Então para qualquer estrutura finita $\mathscr{A}$, o menor ponto fixo de $\varphi^{\mathscr{A}}(R)$ existe. O ponto fixo é igual a um $(\varphi^{\mathscr{A}})^r(\emptyset)$ tal que $r \leq |A|^k$.*

Demonstração. Considere a sequência usando $\varphi^{\mathscr{A}}$ monótono.

$$\emptyset \subseteq \varphi^{\mathscr{A}}(\emptyset) \subseteq (\varphi^{\mathscr{A}})^2(\emptyset) \subseteq ...$$

Se $(\varphi^{\mathscr{A}})^{i+1}(\emptyset)$ contém estritamente $(\varphi^{\mathscr{A}})^i(\emptyset)$ então deve conter pelo menos uma nova tupla de aridade k de A. Já que existe no máximo n^k tuplas então existe um ponto fixo para algum r tal que $r \leq |A|^k$. □

Escrevemos $(LFP_{R^k_{x_1,...,x_k}}\varphi)$ para denotar o menor ponto fixo de $\varphi(R^k,x_1,...,x_k)$. Então o operador (LFP) formaliza a definição de novas relações por indução.

Definição 2.4.5 (Lógica $FO(LFP)$)**.** *$FO(LFP)$ é a linguagem da Lógica de Primeira Ordem adicionada do operador de menor ponto fixo (LFP). Se $\varphi(R^k,x_1,...,x_k)$ é uma fórmula com R^k positivo e x_1, ..., x_k são variáveis livres em φ então $LFP_{R^k x_1...x_k}$ pode ser usado como um novo símbolo relacional de aridade k expressando o menor ponto fixo de φ.*

3 COMPLEXIDADE COMPUTACIONAL

Neste capítulo, vamos apresentar as noções de Complexidade Computacional que vão ser utilizadas no decorrer do trabalho. Na primeira seção, vamos mostrar as definições de máquina de Turing e das classes de complexidade clássicas P, NP, $coNP$ e $NP \cap coNP$. Na segunda seção, apresentaremos as classes de complexidade probabilísticas de tempo polinomial RP, $coRP$, ZPP, BPP e PP.

3.1 Máquinas de Turing e Classes de Complexidade

Nesta primeira seção, vamos introduzir as principais definições e resultados da Complexidade Computacional. Maiores detalhes podem ser encontrados em (PAPADIMITRIOU, 1994; HOPCROFT; MOTWANI; ULLMAN, 2006; ARORA; BARAK, 2009; SIPSER, 2012). Na Teoria da Complexidade Computacional analisamos o custo computacional de resolver um determinado problema. Para formalizar o estudo de problemas em Complexidade Computacional vamos definir a noção de problema de decisão. Para definir essa noção vamos apresentar o problema do Caixeiro-viajante. Nesse problema é dada uma lista de cidades e a distância entre cada par de cidades. O problema consiste em determinar se é possível viajar por todas as cidades, sem repetir nenhuma, deslocando-se apenas por no máximo uma distância k. Temos então apenas duas possíveis soluções para o problema: SIM e NÃO. Problemas computacionais com essa característica são chamados de problemas de decisão. Podemos formular qualquer problema em uma versão de decisão. Utilizamos a versão de decisão para simplificar o estudo dos problemas.

Uma maneira de formalizar um problema de decisão é como uma linguagem. Uma linguagem é apenas um conjunto de sequências de símbolos ("strings" em inglês). Para ver problemas de decisão como linguagens, precisamos codificar as instâncias de um determinado problema como strings. As strings que pertencem à linguagem seriam as codificações das instâncias do problema para as quais a resposta é SIM. Dessa forma, podemos resolver um problema de decisão reconhecendo as strings que pertencem à linguagem correspondente.

Para reconhecer strings em uma linguagem precisamos de um mecanismo que receba a string como entrada e consiga decidir se a string pertence ou não à linguagem que estamos interessados. Máquinas de Turing são esses mecanismos que recebem uma string como entrada, realizam computações na string de entrada e podem retornar um resultado. As definições abaixo podem ser verificadas em (PAPADIMITRIOU, 1994).

A máquina de Turing é constituída de uma fita e de um cabeçote. A computação na máquina de Turing se dá com a movimentação do cabeçote sobre a fita e a modificação dos símbolos escritos em posições dessa fita. O movimento e a alteração da fita são determinados por uma função de transição interna da máquina e pelo símbolo que o cabeçote está lendo na fita. Se a computação da máquina parar, o resultado é dado pelo conteúdo da fita ao fim da computação. Apesar da sua simplicidade, a máquina de Turing pode ser vista como uma linguagem de programação primitiva e um modelo para executar os comandos dessa linguagem

mas, mesmo assim, parece ser capaz de expressar qualquer algoritmo e simular qualquer linguagem de programação. A tese de Church-Turing afirma que esse é o caso. Abaixo vamos formalizar a definição de máquina de Turing:

Definição 3.1.1 (Máquina de Turing Determinística)**.** *Uma máquina de Turing é uma quádrupla* $M = \langle K, \Sigma, \delta, s_0 \rangle$. *K é um conjunto finito de estados e* $s_0 \in K$ *é o estado inicial.* Σ *é um conjunto finito de símbolos, também chamado de alfabeto de M. Os símbolos especiais b e* $\triangleright$ *sempre pertencem à* Σ. *A função* $\delta : K \times \Sigma \to (K \cup \{s_f, yes, no\} \times \Sigma \times \{\leftarrow, \rightarrow, \_\})$ *é a função de transição da máquina onde* s_f*, yes e no são os estados finais, as setas indicam para onde o cabeçote irá se mover e o _ indica que o cabeçote vai ficar na mesma posição. Logo, a função de transição recebe um estado e um símbolo que está sendo lido na fita e retorna um estado, um símbolo que vai ser escrito e uma direção para o cabeçote.*

Para exemplificar a função δ, considere uma transição $\delta(q_1, \alpha) = (q_2, \beta, D)$. A transição nos diz que se a máquina estiver no estado q_1 e lendo o símbolo α na fita então q_2 é o próximo estado, o símbolo β vai ser escrito em cima do símbolo α na fita e o cabeçote será movimentado na direção indicada por $D \in \{\leftarrow, \rightarrow, \_\}$. Se o símbolo da fita que estiver sendo lido for o $\triangleright$ então é assumida a transição $\delta(q_1, \triangleright) = (q_2, \triangleright, \rightarrow)$ pois o $\triangleright$ representa o início da fita e não pode ser apagado.

A execução da máquina começa no estado inicial s_0 e o cabeçote aponta para a primeira posição que é sempre $\triangleright$ da string de entrada. A string de entrada é constituída de símbolos do alfabeto Σ. A partir da configuração inicial, a máquina executa de acordo com a função de transição δ. Já que δ é uma função completamente especificada, só existe uma forma da máquina não poder continuar com a computação. O motivo é se um dos estados finais s_f, *yes*, *no* for alcançado. Quando isso acontece nós dizemos que a máquina para. Se o estado final quando a máquina para for o *yes* então dizemos que a máquina de Turing aceitou a string de entrada. Se for o estado *no* então ela rejeita a entrada. Vamos denotar a saída de uma máquina de Turing M que para com entrada w por $M(w)$. Dessa forma, se o estado final for *yes* ou *no* então dizemos que $M(w) = yes$ ou $M(w) = no$, respectivamente. Se o estado final for s_f então $M(w)$ é a string na fita no final da execução da máquina. É possível que a execução da máquina entre em um ciclo e ela nunca pare. Nesse caso usaremos a notação $M(w) = \uparrow$.

Podemos definir formalmente a execução de uma máquina de Turing usando a noção de configuração. Intuitivamente, uma configuração descreve completamente o estado atual da computação. É como uma foto da execução da máquina. Formalmente, uma configuração é uma tripla (q, w, v) onde $q \in K$ é o estado atual e w e v são strings na fita. w é a string à esquerda do cabeçote e o símbolo sendo lido pelo mesmo. v é a string à direita do cabeçote. A configuração incial da máquina seria $(s_0, \triangleright, w)$ pois o estado inicial s_0 é o estado atual, o cabeçote está lendo o $\triangleright$ e w que é a string de entrada está à direita do cabeçote. A configuração final é a tripla (q_f, w_n, v_n) tal que $q_f \in \{s_f, yes, no\}$ e w_n e v_n são strings quaisquer. Ou seja, a máquina chegou a um estado final.

Definição 3.1.2. *Seja uma máquina de Turing M. Dizemos que uma configuração* (q, w, v) *produz a configuração* (q', w', v') *em um passo se quando M está com configuração* (q, w, v) *a aplicação da função* δ *resulta na configuração* (q', w', v'). *Para denotar essa noção usamos*

a notação $(q,w,v) \rightarrow^{M} (q',w',v')$. *Generalizando, podemos definir quando uma configuração acarreta em outra em k passos denotando por* $(q,w,v) \rightarrow^{M^k} (q',w',v')$ *e quando acarreta em algum número de passos* $(q,w,v) \rightarrow^{M^*} (q',w',v')$.

Agora podemos definir formalmente a computação, ou execução, de uma máquina de Turing.

Definição 3.1.3 (Computação da Máquina de Turing)**.** *Seja uma máquina de Turing M. A computação de M é uma sequência de configurações* $(q_1,w_1,v_1),(q_2,w_2,v_2),...,(q_n,w_n,v_n),...$ *(que pode ser infinita) tal que* (q_1,w_1,v_1) *é a configuração inicial e* $(q_1,w_1,v_1) \rightarrow^{M} (q_2,w_2,v_2)$, $(q_2,w_2,v_2) \rightarrow^{M} (q_3,w_3,v_3)$, ..., $(q_{n-1},w_{n-1},v_{n-1}) \rightarrow^{M} (q_n,w_n,v_n)$ *até não poder mais aplicar a função* δ. *A execução para quando* (q_n,w_n,v_n) *é uma configuração final. A execução não para quando a sequência de configurações é infinita.*

Agora que já formalizamos a ideia de computação e definimos quando uma máquina de Turing para e aceita ou rejeita uma string podemos definir a noção de uma máquina de Turing decidir e aceitar uma linguagem.

Definição 3.1.4 (*M* decide *L*)**.** *Uma máquina de Turing M decide uma linguagem* $L \subseteq \Sigma^*$ *se M aceita todas as strings* $w \in L$ *e rejeita todas as strings* $v \notin L$. *Ou seja, M decide quem está ou não em L. Se uma linguagem L é decidida por uma máquina de Turing dizemos que L é recursiva.*

Definição 3.1.5 (*M* reconhece *L*)**.** *Uma máquina de Turing M reconhece uma linguagem* $L \subseteq \Sigma^*$ *se* $M(w) =$ *yes para todas as strings* $w \in L$ *e não para para todas as strings* $v \notin L$, *ou seja,* $M(v) = \uparrow$. *Se uma linguagem L é aceita por uma máquina de Turing dizemos que L é recursivamente enumerável.*

Pela tese de Church-Turing a classe das linguagens recursivas corresponde à classe de problemas de decisão que têm algoritmos para resolvê-los. A única diferença é que as instâncias dos problemas estão codificadas em strings.

Agora vamos introduzir uma parte essencial da Complexidade Computacional que é verificar a eficiência dos algoritmos. Podemos ter várias medidas de eficiência como, por exemplo, tempo de execução necessário, espaço requerido, número de instruções da máquina, dentre outras. As mais utilizadas são as duas primeiras pois é importante saber o tempo e o espaço necessários na execução de um programa. Essas duas medidas são importantes pois quando implementamos um programa em um computador queremos saber quanto tempo vai demorar para executá-lo e o quanto de memória ele vai utilizar.

Podemos definir o tempo t necessário por uma máquina de Turing para executar com uma entrada w como sendo o número de configurações da inicial até a final. Se $M(w) = \uparrow$ então o tempo requerido por M com entrada w é infinito. Agora precisamos de uma noção de tempo necessário para qualquer string de entrada da máquina. Dizemos que uma máquina M executa em tempo $f(n)$ se, para qualquer string w, o tempo requerido por M na entrada w é no máximo $f(|w|)$ tal que $|w|$ é o tamanho da string w, ou seja, a função $f(n)$ é um limite de tempo

para a máquina M. Agora suponha que uma linguagem L é decidida por uma máquina de Turing que executa em tempo $f(n)$. Dizemos que $L \in DTIME(f(n))$. $DTIME(f(n))$ é um conjunto de linguagens que são decididas por uma máquina de Turing determinística de tempo $f(n)$. Chamamos $DTIME(f(n))$ de classe de complexidade. Essa classe é um conjunto de linguagens dentre as quais algumas representam problemas de decisão importantes. Todos esses problemas de decisão que estão em uma classe de complexidade compartilham uma propriedade sobre a eficiência com que podem ser resolvidos. O D de $DTIME$ se refere à determinístico pois as transições da máquina de Turing que nós definimos são obtidas através da função δ que para cada configuração só pode proceder de uma única maneira. Agora vamos definir a importante classe de complexidade P que significa tempo de execução polinomial e que define a noção de computação eficiente.

Definição 3.1.6 (Classe P)**.** $P = \bigcup_{c \geq 1} DTIME(n^c)$ *tal que* $c \in \mathbb{N}$.

Um problema conhecido que está nesta classe é o $HORNSAT$. Neste problema, recebemos como entrada uma fórmula no formato de cláusulas de horn e o problema é saber se essa fórmula é satisfatível ou não (HUTH; RYAN, 2004; DOWLING; GALLIER, 1984). Esse problema é bastante importante para a classe P pois é um problema completo para essa classe (COOK; NGUYEN, 2010). Isso quer dizer que além desse problema pertencer à classe P, podemos usar um algoritmo que resolva esse problema para resolver qualquer problema dessa classe. Isso acontece porque podemos reduzir qualquer problema da classe P para esse problema. Claramente, essa redução não pode ter uma complexidade computacional muito alta. A seguir vamos definir formalmente as reduções e em seguida os problemas completos.

Definição 3.1.7 (Redução)**.** *Sejam L_1 e L_2 duas linguagens. L_1 pode ser reduzida para L_2 se existe uma função R de strings em strings computável por uma máquina de Turing determinística em espaço logarítmico tal que para toda entrada x, $x \in L_1$ se e somente se $R(x) \in L_2$. R é chamada de redução de L_1 para L_2.*

Definição 3.1.8 (Problema Completo para uma Classe)**.** *Seja $\mathscr{C}$ uma classe de complexidade e L uma linguagem em $\mathscr{C}$. Dizemos que L é $\mathscr{C}$-completa se qualquer linguagem $L' \in \mathscr{C}$ pode ser reduzida para L.*

Podemos definir uma variação da máquina de Turing para ter uma relação no lugar da função de transição δ. Nesse caso, cada configuração pode produzir mais de uma configuração. Podemos dizer que a computação se ramifica e podemos representá-la como uma árvore. Cada nó é uma configuração e seus filhos são as configurações possíveis a partir do nó pai. Chamamos essa máquina de não-determinística. Abaixo vamos definí-la formalmente:

Definição 3.1.9 (Máquina de Turing Não-Determinística)**.** *Uma máquina de Turing não determinística é uma quádrupla $M = \langle K, \Sigma, \Delta, s_0 \rangle$. K, s_0 e Σ são definidos como na máquina de Turing determinística. Para usar o fato da máquina não-determinística ter várias opções na execução o Δ é uma relação tal que $\Delta \subset K \times \Sigma \times (K \cup \{s_f, yes, no, \} \times \Sigma \times \{\leftarrow, \rightarrow, \_\})$. Ou seja, para cada estado e símbolo temos zero ou mais próximo estados possíveis.*

As definições de configuração, transição entre configurações e execução de uma máquina de Turing não-determinística são análogas às da máquina de Turing determinística trocando a função de transição pela relação de transição. Agora vamos definir quando uma máquina não-determinística aceita uma linguagem para podermos definir as classes de complexidade não-determinísticas.

Definição 3.1.10 (N decide L)**.** *Uma máquina de Turing não-determinística N decide uma linguagem $L \subseteq \Sigma^*$ se para toda string w, $w \in L$ se e somente se $(s_0, \triangleright, w) \rightarrow^{N^*} (yes, u, v)$ para strings u e v quaisquer. Ou seja, se existir uma computação a partir do estado inicial que chegue em um estado de aceitação yes.*

Dizemos que uma máquina não-determinística N executa em tempo $f(n)$ se, para qualquer string w de entrada, o número de transições a partir da inicial não for maior que $f(n)$ em qualquer computação. De forma análoga à classe $DTIME(f(n))$ podemos definir a classe $NTIME(f(n))$ como o conjunto de linguagens que são decididas por uma máquina de Turing não-determinística em tempo $f(n)$. Também de forma análoga à classe P podemos definir a classe NP de linguagens que são decididas por uma máquina de turing não-determinística em tempo polinomial.

Definição 3.1.11 (Classe NP)**.** $NP = \bigcup_{c \geq 1} NTIME(n^c)$ *tal que* $c \in \mathbb{N}$.

Uma das questões centrais da Complexidade Computacional é determinar se $P = NP$. O problema é importante pois se essas classes forem iguais podemos ter algoritmos eficientes para os problemas que são representados por linguagens que estão na classe NP. O problema do caixeiro viajante, definido no início do capítulo, é um exemplo de problema que está na classe NP. Outro problema interessante dessa classe é o SAT. Nesse problema temos uma fórmula proposicional na Forma Normal Conjuntiva, ou CNF, de entrada e queremos saber se existe uma valoração que satisfaz a fórmula. Uma máquina não-determinística pode facilmente "chutar" as valorações em tempo polinomial e testar se valoração satisfaz a fórmula também em tempo polinomial. Da mesma forma que o problema $HORNSAT$ é completo para a classe P, o SAT é completo para a classe NP (PAPADIMITRIOU, 1994; COOK, 1971). Ou seja, além de pertencer à classe, qualquer problema de NP pode ser reduzido para o SAT.

Se NP é a classe de linguagens que tem uma máquina não-determinística tal que existe um ramo de aceitação para strings que pertencem à linguagem então o complemento dessas linguagens possuem máquinas não-determinísticas em que existe um ramo de rejeição para strings que não pertencem à linguagem. Por exemplo, para o problema $\overline{SAT}$ podemos modificar a máquina não-determinística do SAT trocando os estado de aceitação pelos de rejeição e vice-versa. Nessa máquina adaptada temos que existe um ramo de rejeição para fórmulas proposicionais de entrada que não estão em $\overline{SAT}$. Isto ocorre porque o ramo de rejeição representa uma valoração que satisfaz a fórmula proposicional de entrada e, dessa forma, ela não é insatisfatível. Abaixo vamos definir a classe dessas linguagens.

Definição 3.1.12 (Classe $coNP$)**.** *coNP é a classe de linguagens formada pelo complemento das linguagens em NP. Ou seja,* $coNP = \{\bar{L} | L \in NP\}$.

Um exemplo de problema da classe *coNP* é a validade de uma fórmula proposicional. A máquina não-determinística pode "chutar" o valor das valorações e fórmula de entrada não é válida se e somente se existe um ramo de rejeição.

Com as definições das classes *NP* e *coNP*, podemos definir a classe $NP \cap coNP$.

Definição 3.1.13 (Classe $NP \cap coNP$)**.** *A classe $NP \cap coNP$ é formada pelas linguagens que estão em NP e em coNP. As linguagens em $NP \cap coNP$ possuem duas máquinas não-determinísticas, uma com a existência de um ramo de aceitação para strings que pertencem à linguagem e outra com a existência de um ramo de rejeição para strings que não pertencem à linguagem.*

Na próxima seção vamos motivar uma outra variação de máquina de Turing que utiliza probabilidades para fazer computações.

3.2 Máquinas de Turing Probabilísticas e Classes de Complexidade Probabilísticas

Existem vários problemas que podem ser resolvidos usando uma abordagem de probabilidade. Para alguns problemas a versão probabilística do seu algoritmo é mais natural e mais eficiente que a versão determinística. Por exemplo, o problema do Corte Mínimo é resolvido de forma mais eficiente quando utilizamos uma abordagem probabilística (MOTWANI; RAGHAVAN, 1995). Em compensação, o algoritmo mais eficiente pode retornar um resultado errado com uma certa probabilidade. Essa probabilidade do erro pode ser diminuída executando o algoritmo várias vezes. Essa é uma boa motivação para estudar algoritmos probabilísticos. Abaixo vamos introduzir as máquinas de Turing probabilísticas para formalizar o estudo das classes de complexidade probabilísticas. Em termos de computabilidade as duas máquinas computam as mesmas funções (GILL, 1977).

Para estudar algoritmos probabilísticos formalmente podemos introduzir a capacidade de jogar moedas em uma Máquina de Turing como feito em (HOPCROFT; MOTWANI; ULLMAN, 2006) e em (ARORA; BARAK, 2009). Outra forma seria utilizar uma Máquina de Turing não-determinística com uma interpretação diferente de aceitação de uma entrada. Essas duas abordagens são equivalentes. Neste trabalho vamos utilizar a segunda abordagem que é a mesma apresentada em (PAPADIMITRIOU, 1994) e em (ARORA; BARAK, 2009). As definições e resultados desta seção são os mesmos apresentados em (PAPADIMITRIOU, 1994).

Definição 3.2.1 (*N* é precisa)**.** *Seja N uma Máquina de Turing não-determinística limitada polinomialmente no tempo. N é precisa quando todas as computações com entrada x têm o mesmo número de passos que é polinomial em $|x|$ e a cada passo existem exatamente duas escolhas não-determinísticas.*

A máquina definida acima é utilizada na definição da máquina de Turing de Monte Carlo. A máquina de Monte Carlo difere da máquina não-determinística na forma de aceitar uma string.

Definição 3.2.2 (Máquina de Turing polinomial de Monte Carlo)**.** *Seja MC uma máquina de Turing não-determinística limitada polinomialmente no tempo como na definição acima. Seja*

$p(n)$ o polinômio que define o número de passos de todas as computações. Para toda string w: se $w \in L$, então pelo menos metade das $2^{p(|w|)}$ computações com entrada w terminam em estados de aceitação. Se $w \notin L$, então todas as computações terminam em estados de rejeição. A máquina MC é uma máquina de Turing polinomial de Monte Carlo.

O número $2^{p(|w|)}$ que representa o total de ramos de computação vem do fato de que todos os ramos têm tamanho $p(|w|)$ e cada passo da execução da máquina *MC* é bifurcado. Pela definição de Máquina de Turing polinomial de Monte Carlo se a saída de uma determinada entrada foi *yes* sabemos com certeza que a string de entrada pertence à linguagem. Isto implica que não há falsos positivos. Já a probabilidade de falsos negativos, ou seja, a resposta da saída ser *no* mas a string de entrada pertencer a linguagem é de no máximo $\frac{1}{2}$.

Em um algoritmo probabilístico sabemos que nem todos os passos da execução são probabilísticos. Já na máquina de Turing probabilística podemos pensar que todos os passos não-determinísticos são o lançamento de uma moeda. Cada escolha do não-determinismo tem uma probabilidade de $\frac{1}{2}$. No caso de instruções não probabilísticas em um algoritmo podemos fazer com que as duas escolhas da máquina sejam iguais. Agora vamos definir a classe das linguagens decididas por uma máquina de Turing polinomial de Monte Carlo.

Definição 3.2.3 (Classe *RP*)**.** *A classe de todas as linguagens que são decididas por máquinas de Turing polinomial de Monte Carlo é chamada de RP (Randomized Polynomial Time).*

Um corte em um grafo é um conjunto de arestas que se forem retiradas deixa o grafo desconectado. Uma aplicação interessante é saber, por exemplo, o corte de menor cardinalidade de um grafo. Em uma rede de computadores, o menor corte seria o menor número de ligações entre os computadores que se todas essas ligações fossem perdidas então teríamos computadores que não poderiam se comunicar dentro dessa rede (MITZENMACHER; UPFAL, 2005).

O problema de saber se existe um corte com apenas uma aresta é um exemplo de problema em *RP*. Vamos chamar o problema de $CUT_{=1}$. Para verificar que $CUT_{=1} \in RP$ basta criar uma máquina não-determinística que verifica não-deterministicamente para cada uma das 2^n partições do conjunto de vértices com cardinalidade n se existe uma única aresta entre elas. Claramente isso pode ser feito em tempo polinomial. Basta verificar se a máquina realmente respeita as restrições da classe *RP*. Claramente, se o grafo não possuir corte com uma aresta todos os ramos da máquina vão ser de rejeição respeitando a segunda condição da classe *RP*. Agora suponha que o grafo tem um corte de uma única aresta (a,b). Para a partição escolhida não ter o corte mencionado os dois vértices a e b devem estar na mesma partição. Ou seja, temos 2^{n-2} casos em que os dois estão na primeira partição e 2^{n-2} em que estão na segunda partição. Dessa forma, um número maior ou igual a metade tem os dois vértices em partições distintas, satisfazendo a primeira restrição da classe *RP*.

A probabilidade de aceitação de uma máquina de Turing polinomial de Monte Carlo não necessita ser a metade. Usando qualquer probabilidade de aceitação p tal que $0 < p < 1$ estritamente entre 0 e 1 conseguimos obter a metade dos estados de aceitação.

Definição 3.2.4 (Classe RP_ε)**.** *A classe de todas as linguagens L que são decididas por máquinas de Turing polinomial não-determinística com a seguinte propriedade: Para toda string w,*

se $w \in L$, então pelo menos uma fração ε das $2^{p(|w|)}$ computações com entrada w terminam em estados de aceitação. Se $w \notin L$, então todas as computações terminam em estados de rejeição.

Teorema 3.2.1. *Para todo ε tal que $0 < \varepsilon < 1$ temos que $RP_\varepsilon = RP$*

Demonstração. ($RP_\varepsilon \subseteq RP$) (Caso 1) $\varepsilon < \frac{1}{2}$. Suponha que a probabilidade de falsos negativos de uma máquina de Monte Carlo M é de no máximo $1-\varepsilon$, onde $\varepsilon < \frac{1}{2}$ e que L seja a linguagem definida por M. Ou seja, para toda string w, se $w \in L$ então pelo menos uma fração ε de todos os ramos é de aceitação e se $w \notin L$ então todos os ramos de M com entrada w são de rejeição. Podemos transformar essa máquina probabilística M em outra M' com probabilidade de falsos negativos de no máximo $\frac{1}{2}$ repetindo a execução um número k de vezes. Repetir significa que em cada estado final executamos novamente a máquina M. Na última repetição, o estado final de um ramo é de aceitação se e somente se pelo menos uma das execuções desse ramo é de aceitação. Na nova máquina M' temos que a probabilidade de falsos negativos é de no máximo $(1-\varepsilon)^k$. Fazendo $k = -\frac{1}{log(1-\varepsilon)}$ temos que essa probabilidade é de no máximo $\frac{1}{2}$. Logo, para a máquina M' temos que para todo w, se $w \in L$ então pelo menos $1-(1-\varepsilon)^k$ $(\frac{1}{2})$ dos ramos da execução de M' com w são de aceitação e se $w \notin L$ então nenhum é de aceitação. O tempo da máquina M' é k vezes o tempo da máquina M que é polinomial. (Caso 2) $\varepsilon \geq \frac{1}{2}$. Se uma máquina M tem as propriedades de RP_ε, ou seja, para toda string w, se $w \in L$ então executando M com entrada w pelo menos uma fração ε de todos os ramos é de aceitação e se $w \notin L$ então todos os ramos de M com entrada w são de rejeição. Claramente, se pelo menos ε dos ramos são de aceitação para $\varepsilon \geq \frac{1}{2}$ então a máquina M também respeita as condições de RP.

($RP \subseteq RP_\varepsilon$) (Caso 1) $\varepsilon \leq \frac{1}{2}$. Análogo ao Caso 2 acima. (Caso 2) $\varepsilon > \frac{1}{2}$. Análogo ao Caso 1 acima. A máquina M' resultante tem probabilidade de falsos negativos de no máximo $(\frac{1}{2})^k$ e basta fazer $k = -\frac{1}{log_\varepsilon(\frac{1}{2})}$. □

Veja que pela prova acima ε não necessita ser uma constante. Qualquer inverso de um polinômio dependente do tamanho da entrada pode ser usado como ε.

Com esse resultado fica mais fácil verificar outros problemas que estão na classe RP. Vamos mostrar que uma versão mais geral do problema apresentado anteriormente também pertence a essa classe. Seja o problema $MINCUT_{\leq K}$ o problema de saber se o corte mínimo de uma grafo de entrada é menor ou igual à um certo K também dado como entrada do problema. O algoritmo para resolver esse problema é o seguinte: escolha uma aresta aleatoriamente e junte os dois vértices das pontas. Por juntar, queremos dizer remover a aresta estre eles e considerar os dois vértices como um só. Em alguns casos, vamos ter arestas múltiplas entre pares de vértices depois dessa operação. Esse processo é repetido até sobrarem apenas 2 vértices. Se existir um número menor ou igual a K de arestas entre os dois vértices o algoritmo retorna sim como resposta, caso contrário retorna não. Para mostrar que esse algoritmo respeita as condições de RP vamos supor primeiro que o corte mínimo do grafo é maior que K. Nesse caso, o algoritmo acima sempre retorna não pois o algoritmo só retorna cortes maiores ou iguais ao corte mínimo. Agora suponha um grafo com corte mínimo de tamanho k menor ou igual ao K de entrada. Seja C o corte mínimo de tamanho k. O grafo tem que ter pelo menos $\frac{kn}{2}$ arestas senão existiria um vértice com grau menor que k e, assim, teríamos um corte com tamanho menor que k. A

probabilidade de nenhuma aresta do corte C ser escolhida em todos os passos é maior ou igual a $\frac{2}{n(n-1)}$. Dessa forma, $MINCUT_{\leq K} \in RP_{\frac{2}{n(n-1)}}$ e pelo teorema acima $MINCUT_{\leq K} \in RP$.

Podemos verificar que a classe RP está entre as classes P e NP. Abaixo vamos mostrar a prova de que $RP \subseteq NP$. Para entender essa relação basta ver a máquina probabilística de Monte Carlo como um caso particular da máquina não-determinística.

Teorema 3.2.2 ($RP \subseteq NP$)**.** *A classe de linguagens decididas por máquinas de Turing de Monte Carlo está contida na classe de linguagens decididas por máquinas de Turing não-determinísticas de tempo polinomial.*

Demonstração. Considere uma linguagem qualquer L decidida por uma máquina de Turing polinomial de Monte Carlo MC. Para uma máquina não-determinística aceitar uma string de entrada basta existir um ramo de aceitação na árvore de computação. Logo, se existe uma máquina probabilística polinomial de Monte Carlo MC que decide L então existe uma máquina não-determinística N que define a mesma Linguagem L. Basta fazer $N = MC$ pois se metade das computações são de aceitação então existe um ramo de aceitação. Dessa forma, toda string w que é aceita pela máquina MC também é aceita pela Máquina N. □

Agora vamos mostrar a prova de que $P \subseteq RP$. A intuição é que uma máquina de Turing polinomial é um caso particular de máquina de Turing não-determinística.

Teorema 3.2.3 ($P \subseteq RP$)**.** *A classe de linguagens decididas por máquinas de Turing determinísticas de tempo polinomial está contida na classe de linguagens decididas por máquinas de Turing de Monte Carlo.*

Demonstração. Considere uma linguagem qualquer L decidida por uma máquina de Turing determinística polinomial M. Seja a máquina de Turing polinomial de Monte Carlo MC tal que as duas escolhas não-determinísticas são idênticas e definidas pela função δ da máquina M. Logo, a máquina MC decide a linguagem L pois se $w \in L$ então a máquina MC aceita w pois todos as computações levam à estados de aceitação. Se $w \notin L$ então a máquina MC rejeita w em todas as folhas da árvore de computação. □

A classe RP é diferente das classes apresentadas anteriormente P e NP pois nem toda máquina não-determinística de tempo polinomial define um problema em RP. Toda máquina de Turing determinística de tempo polinomial decide um problema em P e toda máquina não-determinística de tempo polinomial decide um problema em NP. Para uma máquina não-determinística N definir um problema em RP então para qualquer entrada, ou N não tem nenhum ramo de aceitação ou a maioria dos ramos é de aceitação. Se uma máquina não-determinística para alguma entrada se comportar de forma diferente então ela não define um problema da classe RP.

Dada uma máquina não-determinística de tempo polinomial, não existe uma maneira fácil de descobrir se a máquina define algum problema de RP. Intuitivamente, teríamos que testar a máquina com todas as strings para verificar se ela define algum problema da classe RP. De fato, esse problema é indecidível como vamos ver a seguir.

Teorema 3.2.4. *Seja M uma máquina não-determinística de tempo polinomial. O problema de saber se para todas as entradas, ou todos os ramos de computação rejeitam ou pelo menos a metade aceita é indecidível. Ou seja, é indecidível saber se M decide um problema em RP.*

Demonstração. Podemos supor que o problema é decidível e reduzir o problema $\overline{HALT}$ para ele. Dessa forma, o $\overline{HALT}$ também seria decidível e aí teríamos um absurdo. Seja uma máquina M_1 e uma string x que são entrada do problema $\overline{HALT}$. Vamos construir uma máquina M_2 que com entrada y, primeiro escreve x na fita e dá dois passos não determinísticos. Em um dos ramos, simula M_1 com entrada x por $|y|$ passos. Nos outros 3 ramos, depois de $|y|$ passos a máquina M_2 rejeita. Na simulação de M_1, M_2 aceita se a máquina M_1 parar em até $|y|$ passos, caso contrário rejeita. Suponha que $M_1, x \in \overline{HALT}$. Logo, M_1 não para com x e, dessa forma, depois de $|y|$ passos, para qualquer y, a máquina M_2 rejeita em todos os ramos. Logo, M_2 é uma máquina com as propriedades de RP. Suponha que $M_1, x \notin \overline{HALT}$. Temos que M_1 para com x e, dessa forma, para algum valor de y a máquina M_2 vai aceitar em apenas um quarto dos ramos. Dessa forma, M_2 não tem a propriedade de RP. Com isso, $\overline{HALT}$ é decidível. Isso é um absurdo. □

Esse resultado também vale para outras classes como a $NP \cap coNP$ apresentada na seção anterior e também para as classes probabilísticas que vamos mostrar a seguir com exceção da classe PP. Chamamos classes com essas propriedades de classes semânticas pois dependemos da semântica das máquinas para sabermos se ela define algum problema da classe. Outra forma de provar o resultado anterior para RP e para as demais classes é utilizando o teorema de Rice.

Tendo definido a classe de complexidade probabilística RP podemos definir a classe de linguagens que são complementos das linguagens de RP.

Definição 3.2.5 (Classe $coRP$)**.** $coRP = \{\bar{L} | L \in RP\}$.

Na classe $coRP$ temos as linguagens que são o complemento das linguagens que estão em RP. Ou seja, $L \in RP$ se e somente se $\bar{L} \in coRP$. Usando a definição da classe RP temos para uma linguagem $L \in RP$ que se $w \in \bar{L}$ então $w \notin L$ e então todas as computações da máquina que decide a linguagem que pertence a RP terminam em estados de rejeição. Se $w \notin \bar{L}$ então $w \in L$ e então pelo menos metade das $2^{p(|w|)}$ computações com entrada w terminam em estados de aceitação na máquina probabilística que decide a linguagem de RP. Podemos criar uma Máquina de Turing de Monte Carlo M' para uma linguagem $\bar{L}$ a partir de uma máquina M que decide a linguagem L. Se na máquina M a saída é Sim então na máquina M' a saída é NÃO e se na máquina M a saída é NÃO então na saída da máquina M' a saída é Sim. Dessa forma, temos falsos positivos no lugar de falsos negativos, ou seja, se para uma string w a saída da máquina for NÃO sabemos com certeza que a string não pertence a linguagem mas se a saída for Sim só temos uma probabilidade de acerto maior que $\frac{1}{2}$.

Depois de definir as classes RP e $coRP$ podemos definir a classe que é interseção das duas $RP \cap coRP$. Abaixo vamos definir essa classe:

Definição 3.2.6 (Classe *ZPP*)**.** *Uma linguagem $L \in RP \cap coRP$ possui duas máquinas de Turing polinomiais de Monte Carlo, uma que não tem falsos negativos e outra que não tem falsos positivos. Ou seja, podemos executar cada uma das máquinas alternadamente até que uma resposta definitiva possa ser obtida. No caso de uma linguagem dessa classe temos a garantia de que podemos conseguir uma resposta certa para a pertinência de uma string mas não é possível saber quando a execução vai terminar. Nós denotamos a classe $RP \cap coRP$ por ZPP (Zero-Error Probabilistic Polynomial Time).*

Agora vamos definir uma classe de complexidade probabilística que em contrapartida à classe *ZPP* tem falsos negativos e falsos positivos.

Definição 3.2.7 (Classe *BPP*)**.** *A classe BPP (Bounded-Error Probabilistic Polynomial Time) contém todas as linguagens L para as quais existe uma máquina de Turing não-determinística de tempo polinomial N com a seguinte propriedade: para todas as entradas w, se $w \in L$ então pelo menos $\frac{3}{4}$ das computações de N com entrada w são de aceitação. Se $w \notin L$ então pelo menos $\frac{3}{4}$ das computações são de rejeição.*

Ou seja, a máquina *N* aceita ou rejeita uma string por uma "grande" maioria. Podemos verificar que $RP \subseteq BPP$ na prova do teorema baixo:

Teorema 3.2.5 ($RP \subseteq BPP$.)**.** *RP está contida em BPP.*

Demonstração. Seja $L \in RP$. Para mostrar que $L \in BPP$ basta usar a mesma máquina *RP* e executá-la várias vezes caso o estado final seja *no* para garantir que pelo menos $\frac{3}{4}$ das computações de *N* são de aceitação. Se $w \notin L$ a máquina *RP* rejeita em todas as computações e então pelo menos $\frac{3}{4}$ das computações de *BPP* são de rejeição. □

Definição 3.2.8 (Classe BPP_ε)**.** *A classe BPP_ε contém as linguagens L para as quais existe uma máquina de Turing não-determinística polinomialmente limitada N com a seguinte propriedade: para todas as entradas w, se $w \in L$ então pelo menos um fração ε das computações de N com entrada w são de aceitação. Se $w \notin L$ então pelo menos uma fração ε das computações são de rejeição.*

Da mesma forma que para as outras classes probabilísticas, podemos reduzir o erro da resposta repetindo o algoritmo de um problema da classe *BPP* várias vezes. Diferente da classe *RP*, o erro é sempre menor que $\frac{1}{2}$. O teorema a seguir pode ser provado usando a ideia de repetir várias vezes a máquina. O número de repetições é descoberto pelo Limitante de Chernoff.

Teorema 3.2.6. *Para todo ε tal que $\frac{1}{2} < \varepsilon < 1$ temos que $BPP_\varepsilon = BPP$*

A variação do problema $MINCUT_{\leq K}$ apresentado anteriormente verificando se o corte mínimo tem tamanho exatamente igual a *K* é um problema em *BPP*. Vamos chamá-lo de $MINCUT_{=K}$. Vamos verificar que o mesmo algoritmo apresentado para o problema $MINCUT_{\leq K}$ é um algoritmo para o $MINCUT_{=K}$. Basta mostrar que nesse caso o algoritmo

satisfaz as restrições da classe *BPP*. No caso em que o tamanho do corte mínimo é igual a K o algoritmo pode retornar corte maiores que o mínimo e errar. No caso em que o tamanho do corte mínimo for menor que K o algoritmo pode retornar um corte com tamanho maior que K. Dessa forma, $MINCUT_{=K} \in BPP_\varepsilon$ para algum ε e assim, $MINCUT_{=K} \in BPP$.

Podemos definir também uma classe de linguagens que têm uma máquina M que aceita uma string por uma maioria. Ou seja, mais da metade das computações de M são de aceitação. Se a string não pertencer à linguagem então a metade ou menos dos ramos de computação são de aceitação. Abaixo vamos definir essa classe.

Definição 3.2.9 (Classe *PP*)**.** *A classe PP (Probabilistic Polynomial Time) contém todas as linguagens L para as quais existe uma máquina de Turing não-determinística polinomialmente limitada N com a seguinte propriedade: para todas as entradas w, $w \in L$ se e somente se mais que $\frac{1}{2}$ das computações de N com entrada w são de aceitação.*

Um problema que está nessa classe é o *MAJSAT*. Nesse problema, a entrada é uma fórmula booleana φ e deve ser retornado se a maioria das 2^n valorações satisfazem φ, onde n é o número de proposições atômicas em φ. Para verificar isso basta construir uma máquina N que não-determinísticamente testa cada uma das 2^n valorações. Cada ramo de execução de N escolhe uma valoração distinta para testar. A maioria dos ramos de M com entrada φ é de aceitação se e somente se $\varphi \in MAJSAT$. Claramente a máquina N tem tempo polinomial.

Note que a classe *PP* é uma classe sintática, diferente das classes probabilísticas que vimos anteriormente pois toda máquina não-determinística de tempo polinomial define alguma linguagem de *PP*. Dessa forma, *PP* tem problemas completos e como exemplo de problema *PP*-completo temos o *MAJSAT* (PAPADIMITRIOU, 1994).

Teorema 3.2.7. *Seja ε racional tal que $0 < \varepsilon < 1$. Dizemos que $L \in PP_\varepsilon$ se existe uma máquina não-determinística M tal que para toda entrada w, $w \in L$ se e somente se pelo menos uma fração ε das computações de M com w são de aceitação. Temos que $PP_\varepsilon = PP$.*

Demonstração. Como definimos as máquinas probabilísticas como sempre tendo dois passos não determinísticos temos $2^{p(n)}$ ramos para uma máquina de tempo polinomial $p(n)$. Dessa forma, podemos considerar que $\varepsilon = a/2^b$ para a e b naturais.($PP_\varepsilon \subseteq PP$) Suponha uma linguagem L tal que $L \in PP_\varepsilon$ e com máquina M. Vamos construir uma máquina M' com as restrições de *PP* da seguinte maneira: Primeiro, M' dá uma passo na execução e não faz nada. Temos dois ramos de execução. No ramo da esquerda, executamos a máquina M, no ramo da direita usamos uma máquina com mesmo quantidade de passos de M e que aceita em exatamente $1-\varepsilon$ dos ramos. Suponha uma entrada $w \in L$, logo, M com entrada w tem uma fração r de ramos de aceitação tal que $r > \varepsilon$. Então a fração de ramos de aceitação de M' com entrada w é $r' = (1-\varepsilon+r)/2$. Como $r > \varepsilon$, então $r' > \frac{1}{2}$. Suponha uma entrada $w \notin L$, logo, M com entrada w tem uma fração r de ramos de aceitação tal que $r \leq \varepsilon$. Então a fração de ramos de aceitação de M' com entrada w é $r' = (1-\varepsilon+r)/2$. Como $r \leq \varepsilon$, então $r' \leq \frac{1}{2}$. Logo, M' tem as propriedades de *PP* e define L. Com isso, temos que $L \in PP$.

($PP \subseteq PP_\varepsilon$) Suponha uma linguagem $L \in PP$ com máquina M. Seja PP_ε tal que $\varepsilon = a/2^b \neq \frac{1}{2}$. Para ε ser fração com $0 < \varepsilon < 1$, a pode assumir os valores naturais entre 1

e $2^b - 1$. Vamos construir uma máquina M' da seguinte forma: Nos $b-1$ primeiros passos da máquina ela não faz nada. Dessa forma, temos 2^{b-1} ramos inicialmente. Se $a < 2^{b-1}$, no primeiro ramos executamos M, nos $a-1$ seguintes executamos uma máquina $M_{ac,rej}$ com mesmo tempo de M em que sempre metade aceita e metade rejeita, nos restantes executamos uma máquina com mesmo tempo de M que sempre rejeita em todos os ramos M_{rej}. Se $a > 2^{b-1}$ então no primeiro executamos M, nos $2^b - a - 1$ seguintes executamos a máquina $M_{ac,rej}$ e no restante executamos uma máquina com mesmo tempo de M mas que sempre aceita em todos os ramos M_{ac}. Seja $w \in L$, M com entrada w tem uma fração r de estados de aceitação tal que $r > \frac{1}{2}$. Se $a < 2^{b-1}$, na máquina M' temos uma execução de M, $a-1$ execuções de $M_{ac,rej}$ e 2^{b-1} execuções de M_{rej}. Podemos dividir os ramos de cada execução de M, $M_{ac,rej}$ e M_{rej} em duas partes tendo ao todo 2^b partes. A fração de ramos de aceitação de M' é $r' > a/2^b$ pois $r > \frac{1}{2}$. Podemos usar o mesmo raciocínio para $a > 2^{b-1}$. No caso em que $w \notin L$ podemos fazer de forma análoga. □

A seguir vamos provar que a classe *PP* engloba a classe *NP*.

Teorema 3.2.8 ($NP \subseteq PP$)**.** *NP está contida em PP.*

Demonstração. Seja uma linguagem $L \in NP$ decidida por uma máquina não-determinística N. Vamos criar uma máquina $N2$ da seguinte forma: $N2$ tem um estado inicial e a partir dele duas escolhas não-determinísticas. A primeira vai para o estado inicial de N, a segunda para uma modificação de N que sempre aceita em todos os ramos. Se uma string $w \in L$ então pelo menos um ramo da máquina N vai ser de aceitação e, dessa forma, mais da metade dos ramos de $N2$ vão ser de aceitação. Se a string $w \notin L$ então menos que a maioria dos ramos vai ser de aceitação. □

Abaixo temos uma figura que relaciona as classes de complexidade probabilísticas e não probabilísticas abordadas neste capítulo. A figura representa alguns dos resultados de (GILL, 1977).

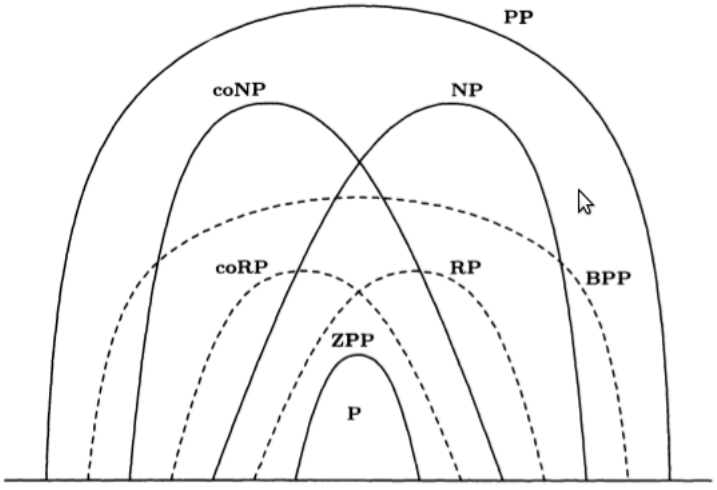

Figura 3.1: Classes de Complexidade

Com isso, apresentamos uma visão geral de Complexidade Computacional. Nos capítulos seguintes vamos apresentar classes de complexidade baseadas em contagem como as classes $\#P$ e $\oplus P$. Algumas ideias deste capítulo vão reaparecer no capítulo seguinte mas como uma outra abordagem.

4 COMPLEXIDADE DESCRITIVA

Nas seções deste capítulo, vamos, primeiramente, apresentar as noções básicas da Complexidade Descritiva e mostrar resultados de caracterização lógica para as classes P e NP. Depois vamos apresentar um resultado de caracterização para a classe de problemas de contagem $\#P$. A prova desse resultado é bastante importante pois a abordagem que utilizamos no nosso trabalho é baseada nela.

4.1 Noções Preliminares e Caracterização das Classes P e NP

A Complexidade Descritiva é uma área da Teoria dos Modelos Finitos (GRäDEL et al., 2005; LIBKIN, 2004) na qual lógicas são utilizadas para caracterizar classes de complexidade. Queremos entender como a expressividade de uma lógica está relacionada com as propriedades de um algoritmo. Enquanto a complexidade computacional está interessada no custo de recursos computacionais como tempo e espaço para decidir se a entrada tem uma certa propriedade, a Complexidade Descritiva se preocupa com os recursos lógicos para definir esta propriedade.

Esta conexão entre complexidade e lógica permite que resultados possam ser facilmente transferidos de uma área para a outra, possibilitando novas abordagens de prova para as duas áreas. Por exemplo, se provarmos que uma mesma lógica caracteriza as classes de complexidade P e NP então temos uma prova de que $P = NP$.

Na Complexidade Descritiva nos concentramos apenas em vocabulários relacionais das lógicas. A maioria das definições e resultados mostrados a seguir podem ser encontrados em (IMMERMAN, 1999). Um vocabulário relacional finito

$$\tau = \left\langle R_1^{a_1}, R_2^{a_2}, ..., R_r^{a_r}, c_1, c_2, ..., c_s \right\rangle$$

é uma tupla finita de símbolos relacionais R_i de aridade a_i e símbolos de constantes. Um exemplo de vocabulário é $\tau_g = \left\langle E^2, f, t \right\rangle$ que representa grafos com um vértice fonte e um terminal.

Uma estrutura $\mathscr{A}$ com vocabulário τ é uma tupla

$$\mathscr{A} = \left\langle A, R_1^{\mathscr{A}}, R_2^{\mathscr{A}}, ..., R_r^{\mathscr{A}}, c_1^{\mathscr{A}}, c_2^{\mathscr{A}}, ..., c_s^{\mathscr{A}} \right\rangle$$

que consiste de um universo não vazio e finito A, o domínio de $\mathscr{A}$, de relações $R_i^{\mathscr{A}}$ de aridade a_i definidas em A^{a_i} para todo símbolo relacional do vocabulário τ, de elementos $c_i^{\mathscr{A}}$ do domínio A para todo símbolo de constante em τ.

Como exemplo temos que o grafo $\mathscr{G} = \left\langle V^{\mathscr{G}}, E^{\mathscr{G}}, 1, 3 \right\rangle$ definido por $V^{\mathscr{G}} = \{0,1,2,3,4\}$ e $E^{\mathscr{G}} = \{(1,2),(3,0),(3,1),\ (3,2),(3,4),(4,0)\}$ é uma estrutura de vocabulário τ_g consistindo de uma grafo direcionado com dois vértices especificados f e t.

Estamos trabalhando com vocabulários e estruturas finitas pois os computadores só manipulam dados finitos tanto na entrada, como na execução e na saída de um programa. Como

vamos usar a lógica para representar os problemas das classes de complexidade computacional precisamos apenas das estruturas finitas. Nós definimos $STRUC[\tau]$ como sendo o conjunto de todas as estruturas finitas com vocabulário τ.

Para qualquer vocabulário τ definimos a linguagem de primeira ordem clássica $\mathscr{L}(\tau)$ consistindo do conjunto de fórmulas como usualmente definido em (EBBINGHAUS; FLUM; THOMAS, 1994) construídas a partir dos símbolos do vocabulário τ, do símbolo de relação $\equiv$, dos conectivos $\wedge$, $\neg$, do quantificador $\exists$ e das variáveis $VAR = \{x, y, z, ...\}$. Como exemplo temos que a fórmula $\exists z((x = z \vee E(x,z)) \wedge (z = y \vee E(z,y)))$ pertence a $\mathscr{L}(\tau_g)$ e significa que existe um caminho de tamanho no máximo 2 entre os vértices x e y. Nesta dissertação nos referiremos à linguagem de primeira ordem como sendo a da Lógica Clássica. Idem para a linguagem de segunda ordem. Veja as seções 2.1 e 2.2 para maiores detalhes. Seja $\phi \in \mathscr{L}(\tau)$, denotamos por $Modelos(\phi)$ o conjunto das estruturas de vocabulário τ que satisfazem a fórmula ϕ.

Quando usamos um computador para armazenar ou manipular uma estrutura, um grafo por exemplo, é codificado de alguma forma uma ordem entre os vértices. Se representarmos o grafo como uma matriz então a ordem dos vértices é dada pela ordem das colunas da matriz. Como queremos usar a lógica para discutir sobre computação é necessário assumir uma ordem no domínio da estrutura. Quando codificamos uma entrada de um programa como uma string existe uma ordem nessa string. Sempre que nos referirmos a um vocabulário τ estaremos assumindo que os símbolos $\leq$, *suc*, *min*, *max* se referem a uma ordem total no domínio, à relação de sucessor e ao menor e maior elementos de um domínio, repectivamente. Para uma estrutura com domínio A tal que $|A| = n$, podemos assumir que $A = \{0, 1, 2, ..., n-1\}$ e que a relação total de ordem $\leq$ é a ordem dos naturais.

Agora vamos definir as consultas. Elas vão ser usadas como os problemas e linguagens da complexidade computacional. As consultas são inspiradas nas consultas de um banco de dados e as estruturas relacionais finitas vão ser os bancos de dados que são as entradas das consultas. Da mesma forma que em banco de dados, as consultas vão retornar uma outra estrutura relacional como resultado da consulta. Ou seja, a entrada e a saída das consultas vão ser estruturas relacionais finitas.

Definição 4.1.1 (Consultas)**.** *Uma consulta Q é um mapeamento $Q : STRUC[\sigma] \to STRUC[\tau]$ das estruturas finitas de um vocabulário σ para estruturas finitas de outro vocabulário τ. Uma consulta booleana Q_b é uma mapeamento $Q_b = STRUC[\sigma] \to \{0,1\}$. Uma consulta booleana pode ser pensada como um subconjunto de $STRUC[\sigma]$, ou seja, o conjunto de estruturas $\mathscr{A}$ para as quais $Q_b(\mathscr{A}) = 1$.*

As consultas de primeira ordem são os tipos mais básicos de consulta. Por exemplo, qualquer fórmula sem variáveis livres (sentença) de primeira ordem $\phi \in \mathscr{L}(\tau)$ define uma consulta booleana Q_ϕ como vamos verificar na definição abaixo:

Definição 4.1.2 (Consultas Booleanas)**.** *Seja uma sentença $\phi \in \mathscr{L}(\tau)$. A consulta Q_ϕ é uma consulta definível por uma sentença, ou simplesmente consulta booleana, sobre o conjunto de estruturas $STRUC[\tau]$ onde $Q_\phi(\mathscr{A}) = 1$ se e somente se $\mathscr{A} \models \phi$.*

Como os problemas de decisão são o padrão para definir classes de complexidades, as consultas booleanas terão um papel central.

Exemplo 4.1.1. *Seja o vocabulário τ_g dos grafos e a sentença $\phi = \forall x \exists y \exists z (y \neq z \wedge E(x,y) \wedge E(x,z) \wedge \forall w (E(x,w) \rightarrow (w = y \vee w = z)))$. Essa sentença define a consulta booleana $Q_\phi : STRUC[\tau_g] \rightarrow \{0,1\}$ de forma que $Q_\phi(\mathscr{G}) = 1$ se e somente se $\mathscr{G}$ for um grafo em que todo vértice tem exatamente duas arestas saindo dele.*

No caso de consultas quaisquer temos como resposta da consulta uma estrutura. Esse tipo de consulta é importante para problemas que não são de decisão. No exemplo abaixo temos uma consulta similar à do exemplo acima mas que retorna um conjunto de vértices que têm exatamente duas arestas saindo deles.

Exemplo 4.1.2. *Seja o vocabulário τ_g dos grafos e a fórmula $\psi = \exists y \exists z (y \neq z \wedge E(x,y) \wedge E(x,z) \wedge \forall w (E(x,w) \rightarrow (w = y \vee w = z)))$. Essa fórmula define a consulta $Q_\psi : STRUC[\tau_g] \rightarrow STRUC[\tau_g]$ de forma que $Q_\phi(\mathscr{G}1) = \langle V^{\mathscr{G}2}, E^{\mathscr{G}2} \rangle$, onde $V^{\mathscr{G}2} = \{v \in V^{\mathscr{G}1} | (\mathscr{G}1, v/x) \models \psi\}$ e $E^{\mathscr{G}2} = \emptyset$. A nova estrutura $\mathscr{G}2$ representa os vértices que têm grau de saída igual a 2.*

A seguir vamos mostrar como uma fórmula define uma consulta k-ária. Essa consulta mapeia uma estrutura $\mathscr{A}$ para uma estrutura $Q(\mathscr{A})$ que tem domínio B definido por um subconjunto de todas as k-tuplas do domínio A de $\mathscr{A}$. Cada relação da estrutura de saída $R_i^{a_i}$ é um subconjunto de B^{a_i}. As constantes são elementos de B. O domínio, todas as relações e todas as constantes são definidos a partir de uma fórmula de primeira ordem. Por isso essas consultas são chamadas de consultas de primeira ordem.

Definição 4.1.3 (Consultas de Primeira Ordem)**.** *Sejam σ e τ dois vocabulários quaisquer tal que $\tau = \langle R_1^{a_1}, ..., R_r^{a_r}, c_1, ..., c_s \rangle$ e k um natural fixo. A consulta $Q : STRUC[\sigma] \rightarrow STRUC[\tau]$ é definível por uma tupla de $r+s+1$ fórmulas $\varphi_0, \varphi_1, ..., \varphi_r, \psi_1, ..., \psi_s$ da linguagem de primeira ordem $\mathscr{L}(\sigma)$. Para cada estrutura $\mathscr{A} \in STRUC[\sigma]$ essas fórmulas definem uma estrutura $Q(\mathscr{A}) \in STRUC[\tau]$,*

$$Q(\mathscr{A}) = \left\langle B, R_1^{Q(\mathscr{A})}, ..., R_r^{Q(\mathscr{A})}, c_1^{Q(\mathscr{A})}, ..., c_s^{Q(\mathscr{A})} \right\rangle.$$

O domínio de $Q(\mathscr{A})$ é um subconjunto de A^k definível em primeira ordem,

$$B = \{\langle b^1, ..., b^k \rangle | (\mathscr{A}, b^1/x^1, ..., b^k/x^k) \models \varphi_0(x_1, ..., x_k)\}.$$

Cada relação $R_i^{Q(\mathscr{A})}$ é definido da seguinte forma:

$$R_i^{Q(\mathscr{A})} = \{(\langle b_1^1, ..., b_1^k \rangle, ..., \langle b_{a_i}^1, ..., b_{a_i}^k \rangle) | (\mathscr{A}, b_1^1/x_1^1, ..., b_{a_i}^k/x_{a_i}^k) \models \varphi_i(x_1^1, ..., x_{a_i}^k)\}.$$

Cada símbolo de constante é definido por uma fórmula de primeira ordem como um elemento de B,

$$c_i^{Q(\mathscr{A})} = \textit{a unica tupla } \langle b^1,...,b^k\rangle \in B \textit{ tal que } (\mathscr{A}, b^1/x^1,...,b^k/x^k) \models \psi(x^1,...,x^k)\}.$$

Uma consulta de primeira ordem pode ser booleana ou k-ária como vimos acima. Definimos como FO o conjunto de todas as consultas booleanas de primeira ordem e como $Q(FO)$ o conjunto de todas as consultas de primeira ordem.

As classes de complexidade são definidas como conjuntos de linguagens decididas por máquinas de Turing. Como na Complexidade Descritiva queremos relacionar consultas booleanas com máquinas de Turing precisamos codificar as estruturas das consultas como strings.

Definição 4.1.4 (Codificação)**.** *Considere o vocabulário relacional $\tau = \langle R_1^{a_1}, R_2^{a_2}, ..., R_r^{a_r}, c_1, c_2, ..., c_s\rangle$ e $\mathscr{A}$ uma estrutura ordenada de vocabulário τ e domínio $A = \{0,1,2,...,n-1\}$ e cada símbolo relacional R_i é interpretado em $\mathscr{A}$ como um subconjunto de A^{a_i}. Vamos codificar cada relação $R_i^{\mathscr{A}}$ como uma string binária $bin^{\mathscr{A}}(R_i)$ de tamanho n^{a_i} onde 1 na k-ésima posição indica que a k-ésima tupla sob a ordem lexicográfica pertence à relação. No caso das constantes vamos codificar em $bin^{\mathscr{A}}(c_j)$ a codificação em binário do elemento $c_j^{\mathscr{A}}$ de tamanho fixo $\lceil log(n)\rceil$. A codificação binária da estrutura $\mathscr{A}$ é a concatenação da codificação de cada relação e constante, ou seja:*

$$bin(\mathscr{A}) = bin^{\mathscr{A}}(R_1)...bin^{\mathscr{A}}(R_r)bin^{\mathscr{A}}(c_1)...bin^{\mathscr{A}}(c_s).$$

Não precisamos codificar o domínio pois como assumimos que seus elementos são sempre naturais entre 0 e $n-1$ podemos recuperar o valor n a partir da codificação. Usando essa codificação temos uma forma de relacionar consultas booleanas com máquinas de Turing. A próxima definição usa essa codificação para dizer quando uma máquina de Turing computa uma consulta.

Definição 4.1.5 (Máquina de Turing M computa consulta Q)**.** *Seja $Q : STRUC[\tau] \to STRUC[\sigma]$ uma consulta e M uma máquina de Turing. Dizemos que M computa a consulta Q se para toda estrutura $\mathscr{A} \in STRUC[\tau]$ temos que $M(bin(\mathscr{A})) = bin(Q(\mathscr{A}))$.*

Veja que a definição acima também funcionaria para o caso de consultas booleanas. Basta fazer $bin(1) = yes$ e $bin(0) = no$. Com isso, temos uma relação entre máquinas de Turing e consultas. Como vimos no capítulo anterior, máquinas de Turing decidem linguagens que compõem classes de complexidade. Então agora podemos relacionar consultas com classes de complexidade. Abaixo vamos definir quando uma consulta pertence a uma classe de complexidade computacional.

Definição 4.1.6 ($Q_b \in \mathscr{C}$)**.** *Seja Q_b uma consulta booleana, M a máquina de Turing que computa Q_b, τ um vocabulário e $\mathscr{C}$ uma classe de complexidade computacional. A consulta Q_b está em $\mathscr{C}$ se a linguagem $L = \{bin(\mathscr{A}) | \mathscr{A} \in STRUC[\tau]$ e $M(bin(\mathscr{A})) = yes\}$ pertence a $\mathscr{C}$.*

Para uma linguagem L pertencer a uma classe de complexidade C deve existir uma máquina de Turing que decida a linguagem L e que execute em tempo ou espaço determinados pela classe C. Isso e a definição anterior nos levam a uma definição alternativa da pertinência de uma consulta a uma classe.

Definição 4.1.7 ($Q \in \mathscr{C}$)**.** *Seja Q uma consulta e $\mathscr{C}$ uma classe de complexidade. Se existir uma máquina de Turing que computa Q com tempo e espaço limitados pela classe então Q pertence a $\mathscr{C}$.*

Agora que definimos o relacionamento entre consultas e classes de complexidade computacional e como cada fórmula de uma lógica define uma consulta booleana, também podemos relacionar uma lógica a uma classe de complexidade.

Definição 4.1.8 ($\mathscr{L}$ está em $\mathscr{C}$)**.** *Seja $\mathscr{L}$ uma lógica e $\mathscr{C}$ uma classe de complexidade computacional. A lógica $\mathscr{L}$ está em $\mathscr{C}$ se para todo vocabulário τ e sentença $\phi \in \mathscr{L}(\tau)$ a consulta booleana Q_ϕ está em $\mathscr{C}$.*

Agora podemos definir o que significa uma lógica $\mathscr{L}$ caracterizar uma classe de complexidade computacional $\mathscr{C}$.

Definição 4.1.9 ($\mathscr{L}$ captura $\mathscr{C}$)**.** *Seja $\mathscr{L}$ uma lógica e $\mathscr{C}$ uma classe de complexidade computacional. $\mathscr{L}$ captura $\mathscr{C}$ se $\mathscr{L}$ está em $\mathscr{C}$ e se, para toda consulta booleana Q_b que está na classe de complexidade $\mathscr{C}$, existe uma sentença $\phi_{Q_b} \in \mathscr{L}$ tal que $\mathscr{A} \models \phi_{Q_b}$ se e somente se $Q_b(\mathscr{A}) = 1$.*

Abaixo vamos enunciar dois teoremas e comentar a intuição deles a partir das definições que acabamos de mostrar.

Teorema 4.1.1 ($FO \subseteq L$)**.** *. O conjunto de consultas booleanas de primeira ordem está contido no conjunto de consultas computáveis em espaço logarítmico em uma máquina de Turing determinística (IMMERMAN, 1999).*

Demonstração. O teorema enuncia que uma lógica está em uma classe de complexidade computacional. Usando essa definição e a definição alternativa de uma consulta pertencer a uma classe de complexidade podemos provar o teorema construindo uma máquina de Turing de espaço logarítmico que computa qualquer consulta booleana definida em primeira ordem. Seja ϕ uma sentença de primeira ordem sem quantificadores. Como ϕ é uma combinação booleana de fórmulas atômicas basta olhar a codificação para verificar se a estrutura satisfaz a fórmula. Isso pode ser feito com espaço logarítmico. Suponha que para uma constante c e uma fórmula qualquer com $n-1$ quantificadores $\psi(c) = Q_2 x_2 ... Q_n x_n \phi$ existe uma máquina de Turing de espaço logarítmico M_0 que decide se uma estrutura de entrada satisfaz a fórmula. Considere agora a fórmula $\exists x \psi(x)$, trocando a constante c que ocorria em ψ por uma variável livre x. Podemos construir uma máquina espaço logarítmica M que tenta todos os valores possíveis de x trocando-o por uma nova constante e executa a máquina M_0. Para armazenar essa nova constante também só precisamos de espaço logarítmico. Se para algum dos valores a máquina M_0 chega em estado de aceitação então a máquina M aceita, caso contrário rejeita. No caso da fórmula $\forall x \psi(x)$ podemos fazer parecido mas para a máquina M aceitar, todos os valores substituídos por x têm que ser aceitos em M_0. □

A prova do teorema nos dá um algoritmo (máquina de Turing) para computar a checagem de modelo da lógica de primeira ordem. Se quisermos saber se uma fórmula é satisfeita

em uma estrutura basta usar a codificação *bin* para a estrutura e máquina de Turing definida na prova.

O próximo teorema foi provado na tese de Doutorado de Fagin em (FAGIN, 1973). Foi o teorema que deu início à área da Complexidade Descritiva.

Teorema 4.1.2 ($SO\exists = NP$)**.** *. O conjunto de consultas booleanas definidas no fragmento existencial de segunda ordem é igual ao conjunto de consultas computáveis em tempo polinomial em uma máquina de Turing não-determinística. Ou seja, a lógica $SO\exists$ captura a classe de complexidade NP.*

Demonstração. ($SO\exists \subseteq NP$).
Seja $\Phi = \exists X_1 ... \exists X_k \psi$ uma fórmula de $SO\exists$. Basta criar uma máquina não-determinística que "chuta" os valores das variáveis de segunda ordem. Depois, basta verificar, em cada ramo, se a estrutura de entrada adicionada das relações computadas não-deterministicamente satisfaz a fórmula ψ. O "chute" não-determinístico é feito em tempo polinomial e o último passo é feito em espaço logarítmico pelo Teorema 4.1.1.

($NP \subseteq SO\exists$).
Seja N uma máquina de Turing não-determinística de tempo polinomial, ou seja, usa tempo $n^k - 1$ para entradas $bin(\mathscr{A})$ com $n = |\mathscr{A}|$. Nós vamos mostrar uma sentença da lógica de segunda ordem existencial

$$\Psi = \exists C_1 ... \exists C_g \exists \Delta \Phi$$

que expressa que existe uma computação de aceitação em N. Veja que a fórmula depende do tempo da máquina N. Isso não é problema pois a máquina já é conhecida. Agora vamos mostrar como expressar as computações de N através da fórmula Ψ.

Cada relação $2k$-ária C_j representa se o símbolo j está em uma determinada célula da fita da máquina em um determinado tempo. As relações C_j são $2k$-árias pois só temos os elementos de 0 a $n-1$ e precisamos codificar os tempos e células usando uma codificação n-ária. Por exemplo, o tempo $n^k - 1$ seria representado pela tupla de k elementos $\langle n-1, ..., n-1 \rangle$. Vamos usar a notação $\bar{x}$ para representar tuplas de elementos.

O conjunto finito de símbolos é $\Gamma = \{\gamma_0, ... \gamma_g\} = \Sigma \cup (Q \times \Sigma)$, ou seja, é um símbolo do alfabeto da máquina ou um par estado e símbolo da máquina. O segundo nos diz onde o cabeçote da máquina está posicionado. Por exemplo, se $\gamma_0 = \langle q_0, w_0 \rangle$ e $\langle \bar{0}, \bar{0} \rangle \in C_0$ então no tempo $\bar{0}$ e na célula $\bar{0}$ o cabeçote está na célula $\bar{0}$ lendo o símbolo w_0.

O Δ representa o ramo da computação de aceitação. Se uma tupla $\bar{t} \in \Delta$ então no tempo $\overline{t+1}$ o ramo escolhido é o da esquerda, caso contrário o da direita é o escolhido. Agora, vamos mostrar como a fórmula Φ garante que existe um ramo de aceitação, que dois símbolos diferentes não podem estar na mesma célula no mesmo tempo e que os C_j seguem diretamente dos próprios C_j e do Δ.

Para garantir que a máquina pára em um estado de aceitação nós podemos assumir que quando N aceita uma entrada ela limpa a sua fita, move o cabeçote para a primeira célula e

entra no estado q_f. Seja γ_f o símbolo correspondente ao estado final com o cabeçote na primeira célula. Logo, a fórmula φ_1 que garante a aceitação dos estados finais é:

$$\varphi_1 = C_f(\bar{0}, \overline{max}).$$

Onde $\bar{0}$ e $\bar{max}$ são tuplas de k elementos que em $\bar{0}$ todos são iguais a 0 e em $\overline{max}$ todos iguais a $n-1$.

Precisamos, também, de uma sentença para especificar que no mesmo tempo $\bar{t}$ e em uma mesma célula $\bar{s}$ não é possível ter dois símbolos diferentes. Essa fórmula é definida a seguir:

$$\varphi_2 = \bigwedge_k \bigwedge_{i \neq j} \forall \bar{s} \forall \bar{t} \neg (C_i^k(\bar{s}, \bar{t}) \wedge C_j^k(\bar{s}, \bar{t})).$$

Agora precisamos definir uma fórmula que diz que para todo $\bar{t}$, a pertinência do elemento $\overline{t+1}$ aos predicados C_j segue da pertinência de $\bar{t}$ ao predicado Δ. A próxima sentença φ_3 especifica que o conteúdo da fita na célula e tempo $(\bar{s}, \overline{t+1})$ segue dos conteúdos das células adjacentes no tempo anterior $(\overline{s-1}, \bar{t})$, $(\bar{s}, \bar{t})$ e $(\overline{s+1}, \bar{t})$ de acordo com os $\Delta(\bar{t})$. Seja $\langle \alpha_{-1}, \alpha_0, \alpha_1, \delta \rangle \rightarrow^N \beta$ uma transição na máquina N onde os conteúdos das células $\alpha_{-1}, \alpha_0, \alpha_1$ levam ao conteúdo β através da transição δ.

$$\varphi_3 = \forall \bar{t}(\bar{t} \neq \overline{max}) \rightarrow \forall \bar{s}(\bar{0} < \bar{s} < \overline{max}) \rightarrow$$
$$\bigwedge_{\langle \alpha_{-1}, \alpha_0, \alpha_1, \delta \rangle \rightarrow^N \beta} (\neg^{\delta} \Delta(\bar{t}) \vee \neg C_{\alpha_{-1}}(\overline{s-1}, \bar{t}) \vee \neg C_{\alpha_0}(\bar{s}, \bar{t}) \vee \neg C_{\alpha_1}(\overline{s+1}, \bar{t}) \vee C_{\beta}(\bar{s}, \overline{t+1})).$$

Onde $\neg^{\delta}$ é $\neg$ se $\delta = 1$ e o símbolo vazio caso contrário.

Logo, $\Phi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3$. Agora temos que mostrar que para toda máquina de Turing não-determinística de tempo polinomial temos uma fórmula Ψ tal que $N(bin(\mathscr{A}))$ tem ramo de aceitação se e somente se $\mathscr{A} \models \Psi$. Suponha uma máquina N tal que $N(bin(\mathscr{A}))$ tem um estado final de aceitação. Pela construção da nossa fórmula Ψ que simula a máquina N, temos que $\mathscr{A} \models \Psi$. Agora suponha que $\mathscr{A} \models \Psi$. Logo, existem predicados que garantem que a máquina de Turing não-determinística N tem uma computação com estados final de aceitação. □

Para entender melhor como podemos utilizar as fórmulas da Lógica Existencial de Segunda Ordem para expressar problemas da classe NP vamos mostrar dois exemplos a seguir:

Exemplo 4.1.3. *O problema SAT recebe como entrada uma fórmula na CNF e a saída deve retornar se essa fórmula é satisfatível. Para usarmos uma estrutura como fórmula vamos usar o vocabulário $\sigma = \{P, N\}$ tal que os dois símbolos representam predicados binários e $P(c_i, p_j)$ expressa que a proposição atômica p_j ocorre positivamente na cláusula c_i. A mesma ideia é usada para o predicado N sendo que proposição atômica ocorre negativamente neste caso. O problema SAT é expressível em SO$\exists$ da seguinte maneira:*

$$\phi_{SAT} = \exists S \forall c \exists p((P(c,p) \wedge S(p)) \vee (N(c,p) \wedge \neg S(p))).$$

A fórmula diz que existe uma valoração S com as variáveis proposicionais que devem ser verdade para a fórmula ser verdadeira.

Exemplo 4.1.4. *O problema NP-completo CLIQUE é o conjunto de pares $\langle G,K\rangle$ tal que o grafo G tem uma clique de tamanho K. O vocabulário é $\tau_{gk} = \{E^2,k\}$. CLIQUE é expressível em SO$\exists$ da seguinte maneira:*

$$\phi_{CLIQUE} = \exists f(\forall x\forall y(f(x) \neq f(y) \rightarrow x \neq y) \wedge \forall x\forall y(f(x) = f(y) \rightarrow x = y) \wedge \forall x\forall y((x \neq y \wedge f(x) < K \wedge f(y) < K) \rightarrow E(x,y))).$$

A fórmula diz que existe uma relação f que é função e é injetiva. Dessa forma, existe uma rotulação dos vértices dada pela função injetiva f de forma que os vértices rotulados de 0 até $k-1$ formam uma clique.

A seguir vamos apresentar mais uma forma de provar resultados de complexidade descritiva. Suponha que sabemos que uma consulta booleana Q é completa para um classe de complexidade $\mathscr{C}$ via reduções de primeira ordem. Suponha ainda que Q pode ser expressa em uma linguagem $\mathscr{L}$ que também é fechada sob reduções de primeira ordem. Segue daí que a linguagem $\mathscr{L}$ pode expressar tudo da classe $\mathscr{C}$.

A seguir temos um novo framework para provar que uma linguagem $\mathscr{L}$ caracteriza uma classe de complexidade $\mathscr{C}$ que vai seguir os seguintes passos:

- Mostrar que $\mathscr{L} \subseteq \mathscr{C}$ produzindo um algoritmo em $\mathscr{C}$ para cada sentença $\varphi \in \mathscr{L}$ que computa a consulta booleana $MOD[\varphi] = \{\mathscr{A} | \mathscr{A} \models \varphi\}$.
- Construir uma consulta booleana T que é completa para $\mathscr{C}$ via reduções de primeira ordem.
- Mostrar que $\mathscr{L}$ é fechada sob reduções de primeira ordem.
- Expressar a consulta booleana T na linguagem $\mathscr{L}$.

A seguir vamos usar o framework acima para provar que a Lógica de Primeira Ordem acrescida do operador de menor ponto fixo descreve exatamente o conjunto de problemas computáveis em tempo polinomial em uma máquina de Turing determinística.

Teorema 4.1.3 ($FO(LFP) = P$)**.** *$FO(LFP)$ caracteriza exatamente a classe P em estruturas finitas ordenadas.*

Prova: ($FO(LFP) \subseteq P$) Seja $\mathscr{A}$ uma estrutura de entrada com $|A| = n$ e $LFP_{Rx_1\ldots x_k}\varphi \in FO(LFP)$. Pelo Teorema 2.4.1, sabemos que até chegar no ponto fixo iteramos $\varphi^{\mathscr{A}}(\emptyset)$ no máximo n^k vezes. Essa quantidade de iterações significa avaliar a fórmula de primeira ordem φ n^k vezes e cada avaliação é feita em tempo polinomial. Logo, $MOD[LFP_{Rx_1\ldots x_k}\varphi] \in P$.

($P \subseteq FO(LFP)$) O problema $HORNSAT$ é completo via reduções de primeira ordem para a classe P. $FO(LFP)$ é fechada sob reduções de primeira ordem. Podemos expressar o problema $HORNSAT$ em $FO(LFP)$ da seguinte maneira:

$$HORNSAT = \forall c \exists x((P(c,x) \wedge LFP\varphi(x)) \vee (N(c,x) \wedge \neg LFP\varphi(x))),$$

onde $\varphi(R,x) = \exists c(P(c,x) \wedge \forall y(N(c,y) \rightarrow R(y)))$.

4.2 Caracterização da Classe de Contagem #*P*

Nesta seção vamos mostrar uma caracterização da classe de funções #*P*. Esta seção é de importância fundamental no nosso trabalho pois as provas dos nossos resultados podem ser vistas como uma adaptação e generalização da prova da caracterização da classe #*P*.

A classe #*P* foi apresentada primeiramente em (VALIANT, 1979). #*P* é a classe de problemas de contagem com uma função de contagem associada à entrada. A função de contagem aplicada em uma entrada é o número de caminhos de aceitação de uma máquina não-determinística executada com essa entrada. Essa classe tem vários problemas derivados de problemas de decisão conhecidos. Como exemplo, podemos citar o #*SAT* que retorna a quantidade de valorações que satisfazem uma fórmula de entrada.

Agora vamos mostrar as definições de problemas de contagem, da classe #*P* e mostrar a caracterização dessa classe. Os resultados e as definições desta seção são adaptações dos encontrados em (SALUJA; SUBRAHMANYAM; THAKUR, 1995). Realizamos adaptações para encaixar nas nossas definições. Abaixo vamos formalizar a noção de problema de contagem.

Definição 4.2.1 (Problema de Contagem)**.** *Um Problema de Contagem é uma tupla* $\mathscr{PC} = \langle \mathscr{I}_{\mathscr{PC}}, \mathscr{S}_{\mathscr{PC}}, \mathscr{F}_{\mathscr{PC}} \rangle$ *tal que* $\mathscr{I}_{\mathscr{PC}}$ *é um conjunto de instâncias de entrada,* $\mathscr{S}_{\mathscr{PC}}(\mathscr{I})$ *é um conjunto de soluções para a entrada* $\mathscr{I}$ *e* $\mathscr{F}_{\mathscr{PC}} : \mathscr{I}_{\mathscr{PC}} \rightarrow \mathbb{N}$ *é uma função de contagem tal que* $\mathscr{F}_{\mathscr{PC}}(\mathscr{I}) = |\mathscr{S}_{\mathscr{PC}}(\mathscr{I})|$.

Agora vamos definir formalmente a classe #*P* definindo os problemas de contagem que pertencem à essa classe.

Definição 4.2.2 (Classe #*P*)**.** *Um problema de contagem* $\mathscr{PC} \in$ #*P se existe uma máquina de Turing não-determinística de tempo polinomial na qual o número de ramos de computação de aceitação para uma entrada* $\mathscr{I}$ *é dado por* $\mathscr{F}_{\mathscr{PC}}(\mathscr{I})$.

De forma semelhante como fizemos com os problemas de decisão na seção 3.1, vamos assumir que as entradas são estruturas de primeira ordem com as mesmas condições anteriores. Dessa forma, a mesma função de codificação vai ser usada para passar uma estrutura como entrada para uma máquina de Turing. A seguir vamos definir a classe #*FO* formada pelos problemas de contagem definíveis com uma sentença da Lógica de Primeira Ordem.

Definição 4.2.3 (Classe #*FO*)**.** *Seja* $\mathscr{A}$ *uma estrutura sobre um vocabulário* σ *e seja* $\bar{S}$ *uma tupla de variáveis de segunda ordem. Podemos dizer que um problema de contagem* $\mathscr{PC} \in$ #*FO se existe uma fórmula de primeira ordem* $\varphi_{\mathscr{PC}}(\bar{S})$ *sobre* $\sigma \cup \bar{S}$ *tal que*

$$\mathscr{F}_{\mathscr{PC}}(\mathscr{A}) = |\{\bar{S} : (\mathscr{A}, \bar{S}) \models \varphi_{\mathscr{PC}}(\bar{S})\}|.$$

Abaixo vamos mostrar dois exemplos de problemas de contagem (#3$CLIQUE$ e #SAT) e as fórmulas que definem os respectivos problemas.

Exemplo 4.2.1. *O problema de contagem #3CLIQUE recebe como entrada um grafo não direcionado e a função de contagem retorna o número de cliques de tamanho três do grafo. Podemos definir a função de contagem e a fórmula como a seguir*

$$\mathscr{F}_{\#3CLIQUE}(\mathscr{G}) = |\{C : (\mathscr{G},C) \models \exists z_1,z_2,z_3 C(z_1) \wedge C(z_2) \wedge C(z_3) \wedge \forall z(C(z) \rightarrow z = z_1 \vee z = z_2 \vee z = z_3) \wedge E(z_1,z_2) \wedge E(z_2,z_3) \wedge E(z_3,z_1) \wedge z1 < z2 \wedge z2 < z3\}|.$$

Note que na fórmula acima temos que usar a ordem entre os elementos para impedir vértices repetidos e que os mesmos três vértices apareçam em mais de uma solução.

Agora vamos mostrar o próximo exemplo.

Exemplo 4.2.2. *O problema de contagem #SAT recebe como entrada uma fórmula na CNF e a função de contagem retorna o número de valorações que satisfazem a fórmula. Para usarmos uma estrutura como fórmula vamos usar o vocabulário $\sigma = \{P,N\}$ tal que os dois símbolos representam predicados binários e $P(c_i, v_j)$ expressa que a variável v_j ocorre positivamente na cláusula c_i. A mesma ideia é usada para o predicado N sendo que a variável ocorre negativamente neste caso. Abaixo vamos definir a função de contagem a partir de uma fórmula*

$$\mathscr{F}_{\#SAT}(\mathscr{A}) = |\{S : (\mathscr{A},S) \models \forall c \exists v((P(c,v) \wedge S(v)) \vee (N(c,v) \wedge \neg S(v)))\}|.$$

A variável de segunda ordem S acima representa a valoração que atribui verdadeiro às variáveis que pertencem ao predicado.

Agora que mostramos exemplos para entender melhor como podemos definir problemas de contagem usando fórmulas de primeira ordem com variáveis de segunda ordem, o próximo passo é mostrar a caracterização da classe #P pela classe de problemas de contagem definidos por fórmulas da Lógica de Primeira Ordem que estamos chamando de #FO.

Teorema 4.2.1 (#$FO = \#P$)**.** *#FO captura a classe de problemas de contagem #P sobre estruturas ordenadas.*

Demonstração. Para provar a ida vamos supor um problema $\mathscr{PC} \in \#FO$. Temos que mostrar que existe uma máquina de tempo polinomial não-determinística N que recebe como entrada uma estrutura $\mathscr{A}$ de $\mathscr{PC}$ e a quantidade de ramos de aceitação é igual a $\mathscr{F}_{\mathscr{PC}}(\mathscr{A})$. Temos que essa máquina existe pois basta a máquina não-determinística N testar não-deterministicamente todos os valores de $\bar{S}$. Podemos fazer isso com o auxílio da máquina M do teorema 10. Logo, temos que

$$\mathscr{F}_{\mathscr{PC}}(\mathscr{I}_{\mathscr{PC}}) = |\{\bar{S} : (\mathscr{A},\bar{S}) \models \varphi_{\mathscr{PC}}(\bar{S})\}| = |\{\bar{S} : M(bin(\mathscr{I}_{\mathscr{PC}},\bar{S})) \text{ aceita }\}|.$$

Logo, $\mathscr{PC} \in \#P$.

Para provar a volta vamos assumir que $\mathscr{PC} \in \#P$. Logo, existe uma máquina não-determinística N tal que o número de ramos de aceitação com uma entrada $\mathscr{A}$ é igual a $\mathscr{F}_{\mathscr{PC}}(\mathscr{A})$. É facil ver que checar se $\mathscr{F}_{\mathscr{PC}}(\mathscr{A})$ é maior que 0 é um problema em NP. Pela prova do teorema de Fagin, temos que existe uma fórmula de segunda ordem existencial $\exists \bar{S} \varphi(\bar{S})$ tal que $\mathscr{F}_{\mathscr{PC}}(\mathscr{A}) > 0$ se e somente se $(\mathscr{A}, \bar{S}) \models \exists \bar{S} \varphi(\bar{S})$. A fórmula φ é de tal forma que toda computação de aceitação na máquina N com entrada $\mathscr{A}$ correponde a um valor de $\bar{S}$ tal que $(\mathscr{A}, \bar{S}) \models \varphi$. Ou seja, o número de computações de aceitação de N que é $\mathscr{F}_{\mathscr{PC}}(\mathscr{A})$ é igual a $|\{\bar{S} : (\mathscr{A}, \bar{S}) \models \varphi(\bar{S})\}|$. Logo, o problema $\mathscr{PC} \in \#FO$. □

A prova acima mostra que contar os valores da tupla $\bar{S}$ dá como resultado o valor de $\mathscr{F}_{\mathscr{PC}}$. Em (SALUJA; SUBRAHMANYAM; THAKUR, 1995) foi considerada uma forma mais genérica que conta os valores da tupla $\langle \bar{S}, \bar{z} \rangle$ na qual $\bar{z}$ é uma tupla de variáveis de primeira ordem que ocorrem livres na fórmula. Para os nossos propósitos vamos precisar apenas da versão com variáveis de segunda ordem. Além das definições e provas mostradas neste capítulo, algumas propriedades de restrições sintáticas da lógica apresentada também são provadas em (SALUJA; SUBRAHMANYAM; THAKUR, 1995). Neste trabalho não vamos apresentar essas provas pois vamos precisar apenas da ideia da caracterização de $\#P$ para mostrar uma caracterização para outras classes.

No próximo capítulo, vamos mostrar um resultado existente sobre a caracterização da classe PP. Em (KONTINEN, 2009) foi provada uma caracterização para a hierarquia de contagem. Nesse trabalho é utilizada uma lógica com um quantificador de segunda ordem significando que a maioria das relações têm uma determinada propriedade. Em seguida, os autores apresentam um corolário de que restrigindo essa lógica com "quantifier rank" no máximo 1 é possível caracterizar a classe PP. No próximo capítulo, vamos apresentar esses resultados existentes e depois vamos mostrar uma prova alternativa para a caracterização de PP que consideramos mais simples. Vamos mostrar que essa prova alternativa pode ser realizada através das ideias da caracterização de $\#P$.

Além disso, também no próximo capítulo, vamos usar a mesma abordagem da nossa prova alternativa para mostrar uma caracterização da classe $\oplus P$. A diferença é que vamos utilizar o quantificador generalizado de segunda ordem de paridade $\oplus$ na lógica utilizada. Além dos resultados de caracterizações lógicas da classes PP e $\oplus P$, vamos apresentar fórmulas das lógicas utilizadas para expressar problemas completos dessas classes.

5 COMPLEXIDADE DESCRITIVA DAS CLASSES SINTÁTICAS *PP* E $\oplus P$

Neste capítulo, inicialmente, vamos mostrar resultados obtidos em (KONTINEN, 2009). A primeira seção mostra alguns resultados de definibilidade para o quantificador $Most^k$. Esses resultados são utilizados na segunda seção em que mostramos a prova de que o fragmento da lógica $FO(Most)$ com apenas uma aplicação do quantificador de segunda ordem *Most* engloba a classe *PP*. Na duas últimas seções deste capítulo temos contribuições desta dissertação. Na penúltima seção, mostramos uma prova alternativa para a caracterização de *PP* e na última utilizamos a mesma abordagem para caracterizar logicamente a classe $\oplus P$ utilizando o fragmento da lógica $FO(\oplus)$ sem aninhamento do quantificador $\oplus$.

5.1 $FO(Most^k)$ e Definibilidade para o Quantificador $Most^k$

Nesta seção, vamos mostrar alguns resultados de definibilidade do quantificador generalizado maioria apresentado no capítulo 2. Esses resultados foram provados e são utilizados na abordagem de (KONTINEN, 2009) para caracterizar a hierarquia de contagem (WAGNER, 1986).

A seguir vamos mostrar que o maioria relativizado pode definir o maioria e que o maioria relativizado consegue definir o quantificador de Resher. Todos esses quantificadores de segunda ordem foram mostrados no capítulo 2.

Proposição 5.1.1. $\models Most^k X\psi \leftrightarrow Most^k_r X, X(X = X, \psi)$.

Demonstração. Suponha uma estrutura $\mathscr{A}$ tal que $\mathscr{A} \models Most^k_r X, X(X = X, \psi)$.
$\mathscr{A} \models Most^k_r X, X(X = X, \psi)$ se e somente se $\langle A, (X = X)^{\mathscr{A}}, \psi^{\mathscr{A}} \rangle \in Most^k_r$. $\langle A, (X = X)^{\mathscr{A}}, \psi^{\mathscr{A}} \rangle \in Most^k_r$ se e somente se $|(X = X)^{\mathscr{A}} \cap \psi^{\mathscr{A}}| > |(X = X)^{\mathscr{A}} - \psi^{\mathscr{A}}|$. Isso acontece se e somente se a quantidade de interpretações de X que satisfaz ψ é maior que a quantidade que não satisfaz pois todas satisfazem $X = X$. Logo, esta quantidade é maior do que a metade que é a interpretação da fórmula da esquerda. □

Proposição 5.1.2. $\models Most^k_r X, Y(\psi, \phi) \leftrightarrow \mathscr{R}^k X, Y(\psi \wedge \phi, \psi \wedge \neg\phi)$.

Demonstração. Seja a estrutura $\mathscr{A}$ tal que $\mathscr{A} \models Most^k_r X, Y(\psi, \phi)$. $\mathscr{A} \models Most^k_r X, Y(\psi, \phi)$ se e somente se $\langle A, \psi^{\mathscr{A}}, \phi^{\mathscr{A}} \rangle \in Most^k_r$ se e somente se $|\psi^{\mathscr{A}} \cap \phi^{\mathscr{A}}| > |\psi^{\mathscr{A}} - \phi^{\mathscr{A}}|$ se e somente se $|(\psi \wedge \phi)^{\mathscr{A}}| > |(\psi \wedge \neg\phi)^{\mathscr{A}}|$ se e somente se $\langle A, (\psi \wedge \phi)^{\mathscr{A}}, (\psi \wedge \neg\phi)^{\mathscr{A}} \rangle \in \mathscr{R}^k$ se e somente se $\mathscr{A} \models \mathscr{R}^k X, Y((\psi \wedge \phi), (\psi \wedge \neg\phi))$. □

A seguir vamos mostrar que o quantificador existencial de segunda ordem k-ário é definível com o quantificador $Most^k_r$.

Proposição 5.1.3. *O quantificador $\exists^2_k$ é definível com o quantificador $Most^k_r$.*

Demonstração. $\exists^2_k Y\,\psi(Y) \leftrightarrow Most^k_r Y, Y(\psi(Y), \psi(Y))$

A fórmula do lado esquerdo expressa que a quantidade de interpretações de Y que satisfaz ψ é maior que $\emptyset$. A do lado direito diz que $|\psi(Y)^{\mathscr{A}} \cap \psi(Y)^{\mathscr{A}}| > |\psi(Y)^{\mathscr{A}} - \psi(Y)^{\mathscr{A}}|$ se e somente se $|\psi(Y)^{\mathscr{A}}| > \emptyset$. Logo, as duas fórmulas são equivalentes. □

Teorema 5.1.1. *Seja $k \geq 2$. O quantificador $\mathscr{R}^{k-1}$ é definível na lógica $FO(Most^k)$.*

Demonstração. Dado duas fórmulas $\psi_1(X)$ e $\psi_2(Y)$ com variáveis de segunda ordem livres e $k-1$-árias. Vamos mostrar como definir

$$\mathscr{R}^{k-1} X, Y(\psi_1(X), \psi_2(Y)).$$

Seja $\mathscr{A}$ uma estrutura satisfazendo $|A| \geq 2$. Vamos definir uma coleção C de relações k-árias contendo exatamente metade da quantidade total usando uma tupla fixa de aridade k, $\bar{b} = (b_1, ..., b_k) \in A^k$:

$$C = \{B \subseteq A^k | \bar{b} \in B\}.$$

A definição a seguir vai ser útil para fazer com que os conjuntos definidos abaixo não tenham elementos em comum.

Para $B \subseteq A^{k-1}$ e $a \in A$, definimos

$$B_a = \{(a, \bar{b}) \in A^k | \bar{b} \in B\}.$$

Ou seja, colocamos um elemento fixo a em cada tupla do conjunto B.

Seja

$$G_i = \{B \subseteq A^{k-1} | \mathscr{A} \models \psi_i(B)\}, \text{ para } i \in \{1, 2\}.$$

A condição para satisfazer $\mathscr{R}^{k-1} X, Y(\psi_1(X), \psi_2(Y))$ é que $|G_1| > |G_2|$. $|G_1| > |G_2|$ se e somente se $|G_1| - |G_2| > 0$ se e somente se $|C| + |G_1^*| - |G_2^*| > 2^{|A|^k - 1}$,

onde $G_1^* = \{B_a | B \in G_1\}$ e $G_2^* = \{\{\bar{b}\} \cup B_a | B \in G_2\}$ para algum $a \in A$ tal que $a \neq b_1$. A condição de $a \neq b_1$ faz com que os elementos de C sejam diferentes dos elementos de G_1^* pois todo elemento $B \in C$ tem uma tupla que começa com b_1 e nenhuma tupla dos elementos $B_a \in G_1^*$ começa com b_1.

Incluir a tupla fixada $\bar{b}$ em cada elemento de G_2^* faz com que $G_1^* \neq G_2^*$ pois essa tupla não aparece em nenhum elemento de G_1^*. Além disso, todos os elementos de G_2^* também são elementos de C, ou seja, $G_2^* \subseteq C$.

Logo, temos que $|C| + |G_1^*| - |G_2^*| > 2^{|A|^k - 1}$ se e somente se $|(C \cup G_1^*) - G_2^*| > 2^{|A|^k - 1}$.

Então temos

$$\models \mathscr{R}^{k-1}X,Y(\psi_1(X),\psi_2(Y)) \leftrightarrow \exists x_1...x_k(x_1 \neq x_2 \wedge Most^k Z((Z(x_1...x_k) \vee \gamma_1(Z)) \wedge \neg\gamma_2(Z))),$$

onde

$$\gamma_1(Z) = \forall \bar{z}(Z(\bar{z}) \rightarrow z_1 = x_2) \wedge \psi_1(X(\bar{y})/Z(x_2,\bar{y}))$$
$$\gamma_2(Z) = \forall \bar{z}(Z(\bar{z}) \rightarrow (z_1 = x_2 \vee \wedge_i z_i = x_i)) \wedge \psi_2(Y(\bar{y})/Z(x_2,\bar{y})).$$

e a variável x_2 não ocorre nas fórmulas ψ_1 e ψ_2. As variáveis $x_1,...x_k$ representam a tupla fixa $\bar{b}$. Fizemos $x_1 \neq x_2$ pois x_2 representa o elemento a adicionado ao início das tuplas. As fórmulas $\gamma_1(Z)$ e $\gamma_2(Z)$ representam os conjuntos G_1^* e G_2^*, repectivamente. □

Na prova acima mostramos que apenas uma aplicação do quantificador $Most^k$ é necessária para expressar o quantificador $\mathscr{R}^{k-1}$.

5.2 Caracterização da Classe *PP*

Em (KONTINEN, 2009) foi mostrado que a extensão de FO pelo quantificador $Most^k$, para k natural diferente de zero, caracteriza exatamente a hierarquia de contagem. Vamos chamar essa lógica de $FO(Most)$:

Definição 5.2.1. *$FO(Most)$ é a lógica de primeira ordem adicionada dos infinitos quantificadores $Most^k$ para k natural, ou seja, $FO(Most^k_{k\in\mathbb{N}})$.*

Nesta seção vamos nos focar apenas em uma parte específica da prova. No trabalho acima temos como corolário que

$$PP = \{Q_\varphi | \varphi \in FO(Most) \text{ e } qr(\varphi) \leq 1\},$$

onde $qr(\varphi)$ denota a quantidade máxima de aninhamento dos quantificadores $Most^k$ na fórmula $\varphi \in FO(Most)$. Ou seja, não estamos levando em consideração os quantificadores de primeira ordem. Q_φ é a consulta booleana definida pela sentença φ.

A seguir vamos mostrar um dos lados do resultado mencionado acima. Em seguida vamos provar os dois lados em uma prova alternativa que utiliza nossa abordagem.

Teorema 5.2.1. *$PP \subseteq \{Q_\varphi | \varphi \in FO(Most)$ e $qr(\varphi) \leq 1\}$*

Demonstração. Pelo Teorema 4.1.2 de Fagin, um ramo de computação de uma máquina não-determinística N, usando tempo n^k para entradas de tamanho n, pode ser codificado usando uma fórmula de primeira ordem. Seja $\phi(X)$ a fórmula que expressa que a relação X codifica uma execução de tempo n^k da máquina N. A fórmula $\phi(X)$ é a mesma da prova do teorema de Fagin retirando a subfórmula que expressa que ao final da computação o estado final é um estado de aceitação. Seja $\psi(Y)$ a fórmula que representa que a relação Y codifica uma execução de aceitação de tempo n^k. A aridade de X e Y é l. As fórmulas ϕ e ψ são construídas de forma que cada ramo da execução de N com entrada $bin(\mathscr{A})$ corresponde a uma única relação l-ária em A^l temos que

$\mathscr{A} \models Most_r^l X,Y(\phi(X),\psi(Y))$ se e somente se N aceita $bin(\mathscr{A})$.

□

A prova acima usa os resultados de definibilidade apresentados na seção anterior. Utilizamos o resultado que diz que $Most_r^l X,Y(\phi(X),\psi(Y))$ pode ser expresso por uma fórmula de $FO(Most)$ tal que $qr(\varphi) = 1$. A seguir, na próxima seção, vamos mostrar uma prova alternativa que não usa esses resultados de definibilidade.

5.3 Uma Prova Alternativa de Caracterização da Classe *PP* e o *MAJSAT*

Nesta seção vamos mostrar uma prova alternativa para o resultado mencionado acima. Para facilitar vamos dar um nome ao fragmento de $FO(Most)$ com fórmulas com quantifier rank menor ou igual a 1.

Definição 5.3.1 (Lógica $FO(Most)_{qr\leq 1}$)**.** *$FO(Most)_{qr\leq 1}$ é o fragmento da lógica $FO(Most)$ tal que $\varphi \in FO(Most)_{qr\leq 1}$ se e somente se $qr(\varphi) \leq 1$.*

Teorema 5.3.1 ($FO(Most)_{qr\leq 1} = PP$)**.** *A Lógica $FO(Most)_{qr\leq 1}$ caracteriza exatamente a classe de complexidade PP na classe de estruturas ordenadas.*

Demonstração. ($FO(Most)_{qr\leq 1} \subseteq PP$).
Temos que provar que para toda fórmula $\varphi \in FO(Most)_{qr\leq 1}$ existe uma máquina não-determinística N tal que para toda estrutura $\mathscr{A}$

$\mathscr{A} \models \varphi$ se e somente se $N(bin(\mathscr{A}))$ aceita na maioria dos ramos.

Seja $\varphi = Most^k R\psi(R) \in FO(Most)_{qr\leq 1}$. Seja $\mathscr{A}$ a estrutura de entrada e $n = |A|$. Podemos construir uma máquina não-determinística N que não-deterministicamente escreve uma string binária de tamanho n^k representando R. Em cada passo, N escolhe não-deterministicamente se escreve 0 ou 1. Depois desse número polinomial de passos, nós temos uma estrutura expandida $(\mathscr{A},R^{\mathscr{A}})$. N deve aceitar se e somente se $(\mathscr{A},R^{\mathscr{A}}) \models \psi(R)$. Como visto anteriormente, isso pode ser feito em tempo polinomial. Temos que mais da metade das computações de $N(bin(\mathscr{A}))$ aceitam se e somente se para mais da metade das interpretações de R temos $(\mathscr{A},R^{\mathscr{A}}) \models \psi(R)$.

($PP \subseteq FO(Most)_{qr\leq 1}$).
Seja N uma máquina não-determinística com tempo $n^k - 1$ que decide algum problema da classe PP com $n = |A|$ e entrada $bin(\mathscr{A})$. Dado uma entrada $bin(\mathscr{A})$, saber se existe um ramo de execução da máquina N com essa entrada é um problema em NP. Pelo Teorema 4.1.2 de Fagin, existe uma fórmula da lógica existencial de segunda ordem $\exists S\varphi(S)$ tal que $N(bin(\mathscr{A}))$ tem um ramo de aceitação se e somente se $\mathscr{A} \models \exists S\varphi(S)$. Pelo mesmo Teorema 4.1.2, sabemos que a aridade de S é $k(2g+1)$, onde g é a quantidade de símbolos mais a quantidade de pares estado, símbolo. Para igualar a quantidade de interpretações de S com a quantidade de ramos de computação de N precisamos de uma máquina alternativa N'. Sabemos que existe uma máquina não-determinística N' equivalente a N com tempo $n^{k(2g+1)}$. A máquina N' é construída

a partir de N continuando a execução por mais $n^{k(2g+1)} - n^k - 1$ passos, onde $n^k - 1$ era a quantidade de passos de N, e terminando no mesmo estado que na máquina N. Ou seja, a máquina N' tem $2^{n^{k(2g+1)}}$ ramos de execução e mantém a proporção de ramos de aceitação e de rejeição. Da mesma forma, a variável relacional S tem $2^{n^{k(2g+1)}}$ interpretações em um domínio com quantidade de elementos n. Cada ramo de execução de N' com entrada $bin(\mathscr{A})$ corresponde a uma interpretação diferente $S^{\mathscr{A}}$ de S. Assim, a maioria dos ramos de N' com entrada $bin(\mathscr{A})$ é de aceitação se e somente se a maioria das interpretações de S satisfazem $\varphi(S)$. Logo,

$\mathscr{A} \models Most^{k(2g+1)} S\varphi(S)$ se e somente se a maioria dos ramos de $N(bin(\mathscr{A}))$ é de aceitação.

□

A prova acima foi baseada na prova de que $\#FO = \#P$. Só tivemos que adaptar a prova utilizando a máquina alternativa N' para garantir que o número de interpretações da relação S fosse igual ao número de ramos da máquina considerada. Na prova de $\#FO = \#P$ apenas o número de interpretações que satisfaziam a fórmula e o número de ramos de aceitação eram necessários serem iguais. No caso acima, em que a proporção é levada em consideração, tivemos que fazer essa adaptação.

Apesar de a prova acima não precisar usar os resultados de definibilidade do quantificador $Most$, esses resultados podem ser interessantes de serem utilizados quando queremos expressar um problema da classe PP através de uma fórmula $\psi \in FO(Most)_{qr\leq 1}$.

Para entender a dificuldade de tentar expressar um problema da classe PP, vamos voltar ao problema $MAJSAT$ apresentado no capítulo 3. Nesse problema, recebemos uma fórmula proposicional φ na CNF e queremos saber se a maioria das 2^n valorações, onde n é o número de fórmulas atômicas de φ, satisfazem φ. O $MAJSAT$ é um problema completo para a classe PP.

Uma forma intuitiva de expressar esse problema seria aproveitar a ideia da representação do problema SAT feita com a Lógica Existencial de Segunda Ordem no Exemplo 4.1.3:

$$Most^1 S^1 \forall c \exists x((P(c,x) \wedge S(x)) \vee (N(c,x) \wedge \neg S(x)))$$

Mas note que utilizamos estruturas com domínio contendo um elemento para cada cláusula e um elemento para cada fórmula atômica. Dessa forma, a variável relacional S tem interpretações com apenas cláusulas como seus elementos. Essas interpretações de S não deveriam contar na quantidade total de valorações da fórmula de entrada. Dessa maneira, mais da metade das valorações podem satisfazer a fórmula φ mas a fórmula acima não ser satisfeita pela estrutura $\mathscr{A}_\varphi$ correspondente.

Para contornar esse problema, vamos adicionar mais um símbolo no vocabulário das fórmulas proposicionais τ_p. O símbolo a ser adicionado é o V representando o conjunto de fórmulas atômicas da fórmula φ.

Com essa definição, podemos representar o problema $MAJSAT$ usando o quantificador $Most_r^k$ da seguinte forma:

$$Most_r^1 S^1, S^1(\forall x(S(x) \rightarrow V(x)), \forall c \exists x((P(c,x) \wedge S(x)) \vee (N(c,x) \wedge \neg S(x)))).$$

Para facilitar, vamos definir $\phi = \forall x(S(x) \rightarrow V(x))$ e $\psi = \forall c \exists x((P(c,x) \wedge S(x)) \vee (N(c,x) \wedge \neg S(x)))$. A fórmula com quantificador $Most_r^1$ definida acima expressa que a quantidade de interpretações de S que satisfazem $\phi \wedge \psi$ é maior que a quantidade que satisfaz $\phi \wedge \neg\psi$. As interpretações de S que satisfazem $\phi \wedge \psi$ representam as valorações que satisfazem a fórmula φ pois apenas fórmulas atômicas pertencem às interpretações de S. A outra fórmula $\phi \wedge \neg\psi$ é satisfeita pelas interpretações de S que são valorações e que não satisfazem ψ. Dessa forma, a quantidade de valorações que satisfazem φ é maior que a quantidade de valorações que não satisfazem φ. Logo, a maioria das valorações satisfazem φ.

Como conseguimos expressar um problema completo da classe PP através da lógica $FO(Most)_{qr\leq 1}$, podemos pensar em uma outra prova que utiliza a mesma abordagem da prova de $P = FO(LFP)$. Basta mostrar que a lógica $FO(Most)_{qr\leq 1}$ é fechada sob reduções de primeira ordem. Vamos deixar isso como trabalho futuro.

5.4 Caracterização da Classe $\oplus P$ Através do Quantificador $\oplus$

Usando nossa abordagem, nesta seção vamos mostrar uma caracterização de $\oplus P$. De modo similar à classe PP, a classe $\oplus P$ pode ser vista como o conjunto de problemas de decisão definidos a partir dos problemas de $\#P$. Um problema em $\oplus P$ pergunta se o bit menos significativo da contagem de todas as soluções é 0 ou 1. O bit é 1 se e somente se o valor retornado no problema de contagem é ímpar. Para caracterizar essa classe vamos estender a lógica de primeira ordem com o quantificador $\oplus^k$ para todo k natural que verifica se um número ímpar de interpretações de uma variável de segunda ordem satisfaz uma determinada propriedade. Abaixo vamos definir a classe $\oplus P$ (paridade P) para depois definir o nosso framework lógico $FO(\oplus)$ para caracterizá-la.

Definição 5.4.1 (Classe $\oplus P$)**.** *Um problema de decisão Q está em $\oplus P$ se existe uma máquina de Turing não-determinística M de tempo polinomial p tal que para toda entrada $\mathscr{A} \in Q$,*

$$\mathscr{A} \in Q \text{ se e somente se } M(bin(\mathscr{A}) \text{ tem um número ímpar de ramos de aceitação.}$$

Um problema completo dessa classe seria o $\oplus SAT$ onde é verificado se uma fórmula tem um número ímpar de valorações que a satisfazem. Note que esse problema é uma versão de decisão do problema $\#SAT$. Basta ver o número retornado no $\#SAT$ em binário. O $\oplus SAT$ retornaria se o último bit é 0 ou 1.

Agora vamos definir a lógica $FO(\oplus)_{qr\leq 1}$.

Definição 5.4.2 (Lógica $FO(\oplus)_{qr\leq 1}$)**.** $FO(\oplus)_{qr\leq 1}$ *é a lógica formada pelas fórmulas $\varphi \in FO((\oplus^k)_{k\in\mathbb{N}})$ tal que $qr(\varphi) \leq 1$, onde qr é a quantidade de aplicações aninhadas do quantificador generalizado de segunda ordem $\oplus$.*

Para facilitar, abaixo vamos definir quando uma estrutura satisfaz uma fórmula $\oplus S\varphi(S)$.

Definição 5.4.3. *Seja $\mathscr{A}$ uma estrutura, S uma variável de segunda ordem e φ uma sentença de primeira ordem. Abaixo vamos definir quando uma estrutura satisfaz $\oplus S\varphi(S)$.*

$$\mathscr{A} \models \oplus S\varphi(S) \text{ se e somente se } |\{S : (\mathscr{A}, S) \models \varphi(S)\}| \text{ é ímpar.}$$

Agora podemos continuar usando as ideias do capítulo anterior para caracterizar a classe $\oplus P$. Primeiro vamos mostrar quando um problema Q é definível com uma fórmula em $FO(\oplus)_{qr\leq 1}$.

Definição 5.4.4. *Seja $\mathscr{A}$ uma estrutura sobre um vocabulário σ e seja S uma variável relacional. Podemos dizer que um problema de decisão Q é definido em $FO(\oplus)_{qr\leq 1}$ se existe uma fórmula $\oplus S\varphi(S)_Q \in FO(\oplus)_{qr\leq 1}$ tal que*

$$\mathscr{A} \in Q \text{ se e somente se } \mathscr{A} \models \oplus S\varphi(S)_Q.$$

Abaixo vamos mostrar um exemplo de como usar a lógica $FO(\oplus)_{qr\leq 1}$ para expressar um problema da classe $\oplus P$. Esse exemplo nos dá uma evidência de que a lógica $FO(\oplus)_{qr\leq 1}$ engloba a classe $\oplus P$.

Exemplo 5.4.1. *O problema de decisão $\oplus SAT$ recebe como entrada uma fórmula na CNF e retorna SIM se e somente se o número de valorações que satisfazem a fórmula é ímpar. Para usarmos uma estrutura como fórmula vamos usar o vocabulário $\sigma = \{P, N, V\}$ tal que os dois primeiros símbolos representam predicados binários e $P(c_i, v_j)$ expressa que a variável v_j ocorre positivamente na cláusula c_i. A mesma ideia é usada para o predicado N sendo que a variável ocorre negativamente neste caso. O predicado unário $V(v_j)$ tem a variável v_j como elemento. Abaixo vamos definir quando uma fórmula representada por uma estrutura $\mathscr{A}$ tem a propriedade do $\oplus SAT$.*

$$\mathscr{A} \in \oplus SAT \text{ se e somente se}$$
$$\mathscr{A} \models \oplus S(\forall x(S(x) \to V(x)) \wedge \forall c \exists v((P(c,v) \wedge S(v)) \vee (N(c,v) \wedge \neg S(v)))).$$

Agora vamos mostrar a prova da caracterização da classe $\oplus P$.

Teorema 5.4.1 ($FO(\oplus)_{qr\leq 1} = \oplus P$)**.** *A classe de problemas $\oplus P$ é caracterizada exatamente pela lógica $FO(\oplus)_{qr\leq 1}$, na classe de estruturas ordenadas.*

Demonstração. ($FO(\oplus)_{qr\leq 1} \subseteq \oplus P$)
Temos que provar que para toda fórmula $\oplus S\varphi(S) \in FO(\oplus)_{qr\leq 1}$ existe uma máquina não-determinística N tal que

$$\mathscr{A} \models \oplus S\varphi(S) \text{ se e somente se } N(bin(\mathscr{A})) \text{ tem um número ímpar de ramos de aceitação.}$$

Seja uma fórmula $\oplus S\varphi(S) \in FO(\oplus)_{qr\leq 1}$. Podemos construir uma máquina não-determinística N com entrada $bin(\mathscr{A})$ que escreve na fita não-deterministicamente os valores possíveis para $S^{\mathscr{A}}$ e depois verifica se $(\mathscr{A}, S^{\mathscr{A}}) \models \varphi(S)$. Pelos resultados do capítulo 3, isso pode ser feito em tempo polinomial. Claramente, temos que

$\mathscr{A} \models \oplus S\varphi(S)$ se e somente se $N(bin(\mathscr{A}))$ tem uma quantidade ímpar de ramos de aceitação.

$(\oplus P \subseteq FO(\oplus)_{qr\leq 1})$

Suponha um problema $Q \in \oplus P$. Logo, existe uma máquina não-determinística M tal que para toda entrada $\mathscr{A} \in Q$ o número de ramos de aceitação de $M(bin(\mathscr{A}))$ é ímpar. Checar se existe um ramo de aceitação em $M(bin(\mathscr{A}))$ é um problema em NP. Logo, pelo Teorema 4.1.2 de Fagin do capítulo 4, temos que existe uma fórmula de segunda ordem existencial $\exists C\varphi(C)$ tal que $M(bin(\mathscr{A}))$ tem ramo de aceitação se e somente se $\mathscr{A} \models \exists C\varphi(C)$. A fórmula φ é construída de forma que todo ramo de computação de aceitação na máquina N corresponde a um valor de C que satisfaz $\varphi(C)$. Logo, o número computações de aceitação de $M(bin(\mathscr{A}))$ é exatamente igual a $|\{C : (\mathscr{A}, C) \models \varphi(C)\}|$. Mas nós sabemos que o número de ramos de aceitação de $M(bin(\mathscr{A}))$ é ímpar, logo, $|\{C : (\mathscr{A}, C) \models \varphi(\bar{S})\}|$ é ímpar. Com isso, temos que

$\mathscr{A} \models \oplus C\varphi(C)$ se e somente se o número de computações de aceitação de $M(bin(\mathscr{A}))$ é ímpar.

□

6 COMPLEXIDADE DESCRITIVA DE CLASSES PROBABILÍSTICAS SEMÂNTICAS DE TEMPO POLINOMIAL E DA CLASSE $NP \cap CONP$

Na primeira seção, vamos mostrar uma caracterização existente da classe probabilística *BPP*. Essa caracterização utiliza uma lógica com semântica probabilística. Nas seções seguintes vamos utilizar a abordagem de quantificadores generalizados de segunda ordem para mostrar uma caracterização alternativa para a classe *BPP* e para capturar as demais classes probabilísticas de tempo polinomial. Acreditamos que a lógica utilizada na nossa caracterização de *BPP* é mais simples que a da caracterização existente. Dessa forma, fica mais fácil expressar problemas. Apesar disso, nosso resultado é para estruturas ordenadas. Também mostramos uma caracterização para a classe $NP \cap coNP$. A nossa abordagem de caracterizações das classes deste capítulo se baseia na prova de $\#FO = \#P$, apresentada no teorema 4.2.1, e na caracterização de *BPP* da primeira seção deste capítulo.

6.1 Caracterização da Classe *BPP*

Um resultado existente apresentando em (EICKMEYER; GROHE, 2010) e em (EICKMEYER, 2011) mostra uma forma de definir lógicas probabilísticas escolhendo as fórmulas de uma dada lógica tal que, para toda estrutura, a fórmula satisfaz essa estrutura em um dado intervalo de probabilidades. Isso acarreta em uma lógica com sintaxe indecidível que fica compatível com o conjunto de máquinas dos problemas em *BPP* que também é indecidível.

No trabalho mencionado, é apresentada a lógica $BPIFP+C$ que é definida de forma semelhante a classe de complexidade *BPP* a partir da lógica do ponto-fixo inflacionário com contagem. IFP é definida de forma similiar à $FO(LFP)$ mas com um operador inflacionário. Um operador $F : \mathscr{P}(B) \to \mathscr{P}(B)$ é inflacionário se $X \subseteq F(X)$ para todo $X \in \mathscr{P}(B)$. IFP é mostrada ser equivalente à $FO(LFP)$ em (GUREVICH; SHELAH, 1985). A lógica $BPIFP+C$ definida caracteriza exatamente a classe *BPP* mesmo em estruturas sem ordem.

A probabilidade é introduzida na lógica permitindo fórmulas de um vocabulário τ usarem um vocabulário maior $\tau \cup \rho$ onde ρ é o conjunto de símbolos de predicados representando os predicados probabilísticos. A seguir vamos apresentar a abordagem de (EICKMEYER; GROHE, 2010). Vamos assumir que o leitor conhece as definições básicas de probabilidade.

Sejam τ e ρ dois vocabulários disjuntos onde ρ é um vocabulário relacional. Os predicados de ρ vão ser tratados como probabilísticos. A seguir, vamos definir uma expansão probabilística de τ-estruturas:

Definição 6.1.1 (Classes de Expansões de um Estrutura)**.** *Para uma estrutura $\mathscr{A}$ de vocabulário τ, seja $P(\mathscr{A},\rho)$ a classe de todas as $\tau \cup \rho$ expansões de $\mathscr{A}$. Cada estrutura pertencente a $P(\mathscr{A},\rho)$ representa uma interpretação diferente para os predicados de ρ. $P(\mathscr{A},\rho)$ pode ser visto como um espaço de probabilidade com distribuição uniforme.*

Vamos utilizar a probabilidade $Pr_{\mathscr{B}\in P(\mathscr{A},\rho)}(\mathscr{B} \models \varphi)$ de que uma expansão de uma τ-estrutura $\mathscr{A}$ satisfaça uma sentença φ para definir lógicas probabilísticas.

Definição 6.1.2 (Lógica $P_{(\alpha,\beta]}L$)**.** *Seja L uma lógica e $0 \leq \alpha \leq \beta \leq 1$. A lógica $P_{(\alpha,\beta]}L$ é definida para cada vocabulário τ como*

$$P_{(\alpha,\beta]}L[\tau] = \bigcup_{\rho}\{\varphi \in L[\tau \cup \rho] | \text{ para toda estrutura } \mathscr{A} \text{ temos } Pr_{\mathscr{B}\in P(\mathscr{A},\rho)}(\mathscr{B} \models \varphi) \leq \alpha \text{ ou } Pr_{\mathscr{B}\in P(\mathscr{A},\rho)}(\mathscr{B} \models \varphi) > \beta\},$$

onde a união "varre" todos os vocabulários ρ disjuntos de τ. O intervalo $(\alpha,\beta]$ é uma lacuna tal que se $\alpha < Pr_{\mathscr{B}\in P(\mathscr{A},\rho)}(\mathscr{B} \models \varphi) \leq \beta$ então $\varphi \notin P_{(\alpha,\beta]}L[\tau]$.

A definição da semântica é feita adicionando o caso:

$$\mathscr{A} \models_{P_{(\alpha,\beta]}L} \varphi \text{ se e somente se } Pr_{\mathscr{B}\in P(\mathscr{A},\rho)}(\mathscr{B} \models_L \varphi) > \beta.$$

Vamos utilizar

$$BPL := P_{(\frac{1}{4},\frac{3}{4}]}L, \text{ onde } L \text{ é uma lógica.}$$

Lógicas com contagem como a $IFP+C$ geralmente são definidas usando estruturas com dois "sorts". O "sort" adicional tem um segmento inicial dos naturais de tamanho apropriado a ser usado na contagem. Entretanto, os resultados obtidos em (EICKMEYER; GROHE, 2010) necessitam apenas de uma contagem limitada obtida através do quantificador de Resher.

Definição 6.1.3 (Lógica $IFP(\mathscr{R})$)**.** *Seja $IFP(\mathscr{R})$ a lógica obtida a partir da Lógica com Ponto-Fixo Inflacionário IFP adicionada de um quantificador generalizado $\mathscr{R}$ representando o quantificador de Resher. Para quaisquer duas fórmulas $\varphi_1(\bar{x})$ e $\varphi_2(\bar{x})$, onde $\bar{x}$ é uma k-tupla de variáveis, nós temos uma nova fórmula:*

$$\mathscr{R}\bar{x}(\varphi_1(\bar{x}), \varphi_2(\bar{x})).$$

A semântica dessa nova fórmula é definida abaixo:

$$\mathscr{A} \models \mathscr{R}\bar{x}(\varphi_1(\bar{x}), \varphi_2(\bar{x})) \text{ se e somente se } |\{\bar{a} \in A^k | \mathscr{A} \models \varphi_1(\bar{a})\}| > |\{\bar{a} \in A^k | \mathscr{A} \models \varphi_2(\bar{a})\}|$$

Teorema 6.1.1 ($BPIFP(\mathscr{R}) = BPP$)**.** *A lógica $BPIFP(\mathscr{R}) = P_{(\frac{1}{4},\frac{3}{4}]}IFP(\mathscr{R})$ captura exatamente BPP na classe de estruturas ordenadas.*

Demonstração. ($BPIFP(\mathscr{R}) \subseteq BPP$) Uma máquina da classe BPP pode chutar as relações probabilísticas.

($BPP \subseteq BPIFP(\mathscr{R})$) Seja Q um conjunto de estruturas da classe BPP. Existe uma máquina não-determinística M que decide a consulta $Q_{\leq}$ de expansões ordenadas das estruturas de Q. Usando o Teorema de Immerman-Vardi segue que existe uma sentença de $BPIFP$ φ_M definindo $Q_{\leq}$.

A lógica $IFP(\mathscr{R})$ é menos expressiva que a lógica $IFP+C$ e $BPIFP+C \subseteq BPP$. Logo, $BPIFP+C = BPIFP(\mathscr{R})$ e ambas caracterizam BPP. □

6.2 Caracterização de *BPP* com Quantificadores Generalizados

Nesta seção vamos utilizar as ideias anteriores para caracterizar a classe *BPP* com outra abordagem. A abordagem que vai ser utilizada é baseada em quantificadores generalizados de segunda ordem e na forma semântica de definir fórmulas de uma lógica vista na caracterização anterior. O interessante dessa caracterização é a alternativa com relação ao uso das lógicas probabilísticas $BPIFP(\mathscr{R})$ e $BPIFP+C$ utilizadas anteriormente. Além de não usar probabilidade na semântica, acreditamos que a nossa alternativa é mais simples para descrever as fórmulas pois a lógica usada anteriormente era a lógica com ponto fixo inflacionário e no nosso caso utilizamos a lógica de primeira ordem adicionada do quantificador $Q^k_{\geq \frac{3}{4}}$ definido no Exemplo 2.3.2. Nas seções seguintes, vamos usar essa mesma abordagem para caracterizar outras classes probabilísticas semânticas que não têm caracterização na literatura e como consequência também vamos caracterizar uma classe semântica que não é probabilística.

Definição 6.2.1 (Lógica $BPFO(Q_{\geq \frac{3}{4}})$)**.** *Seja a lógica* $FO(Q_{\geq \frac{3}{4}}, Q_{<\frac{1}{4}}) = FO((Q^k_{\geq \frac{3}{4}})_{k\in\mathbb{N}}, (Q^k_{<\frac{1}{4}})_{k\in\mathbb{N}})$ *com os quantificadores de segunda ordem definidos como visto anteriormente. Vamos definir o conjunto de fórmulas da lógica* $BPFO(Q_{\geq \frac{3}{4}})$ *como sendo*

$$BPFO(Q_{\geq \frac{3}{4}}) = \{Q_{\geq \frac{3}{4}} R\varphi(R) | \varphi(R) \in FO \text{ e para toda estrutura } \mathscr{A} \text{ temos } \mathscr{A} \models Q_{\geq \frac{3}{4}} R\varphi(R) \text{ ou } \mathscr{A} \models Q_{<\frac{1}{4}} R\varphi(R)\}.$$

Dessa forma, as fórmulas da lógica $BPFO(Q_{\geq \frac{3}{4}})$ têm apenas um quantificador de segunda ordem seguido de uma fórmula da Lógica de Primeira Ordem. Utilizamos os quantificadores $(Q^k_{\geq \frac{3}{4}})_{k\in\mathbb{N}}$ e $(Q^k_{<\frac{1}{4}})_{k\in\mathbb{N}}$ para as fórmulas da lógica sejam definidas de forma análoga às máquinas que definem problemas da classe *BPP*.

A semântica é definida a seguir:

Definição 6.2.2. *Seja* $\mathscr{A}$ *uma estrutura finita e* $Q_{\geq \frac{3}{4}} R\varphi(R) \in BPFO(Q_{\geq \frac{3}{4}})$ *tal que R tenha aridade k.*

$$\mathscr{A} \models Q_{\geq \frac{3}{4}} R\varphi(R) \text{ se e somente se } |\{R|(\mathscr{A}, R) \models \varphi(R)\}| \geq \tfrac{3}{4} \times 2^{|A|^k}.$$

Agora vamos mostrar como caracterizar a classe *BPP* através da lógica $BPFO(Q_{\geq \frac{3}{4}})$ usando a abordagem da caracterização de #*P* vista no Teorema 4.2.1.

Teorema 6.2.1 ($BPFO(Q_{\geq \frac{3}{4}}) = BPP$)**.** $BPFO(Q_{\geq \frac{3}{4}})$ *captura exatamente BPP na classe de estruturas ordenadas.*

Demonstração. ($BPFO(Q_{\geq \frac{3}{4}}) \subseteq BPP$).
Seja uma fórmula $Q_{\geq \frac{3}{4}} R\varphi(R) \in BPFO(Q_{\geq \frac{3}{4}})$ tal que R tenha aridade k. Podemos construir uma máquina não-determinística N que "chuta" os valores da relação R e depois decide se $(\mathscr{A}, R) \models \varphi(R)$, onde $\mathscr{A}$ é uma estrutura de entrada. Como $Q_{\geq \frac{3}{4}} R\varphi(R)$ é uma fórmula de $BPFO(Q_{\geq \frac{3}{4}})$ então temos que para toda estrutura de entrada $\mathscr{A}$, $|\{R|(\mathscr{A}, R) \models \varphi(R)\}| \geq \frac{3}{4} \times$

$2^{|A|^k}$ ou $|\{R|(\mathscr{A},R) \models \varphi(R)\}| < \frac{1}{4} \times 2^{|A|^k}$. Logo, a máquina N é uma máquina com as restrições da classe BPP.

$(BPP \subseteq BPFO(Q_{\geq \frac{3}{4}}))$.

Seja uma máquina N com tempo n^k e com as restrições da classe BPP. Dada uma estrutura de entrada $\mathscr{A}$ saber se N tem um ramo de aceitação com entrada $\mathscr{A}$ é um problema em NP. Pelo teorema de Fagin, temos que existe uma fórmula da lógica existencial de segunda ordem $\exists S\varphi(S)$ tal que $\mathscr{A} \models \exists S\varphi(S)$ se e somente se $N(bin(\mathscr{A}))$ tem um ramo de aceitação. Como visto anteriormente, sempre podemos ajustar a máquina N de forma que a quantidade de ramos de N vai ser igual a quantidade de valores distintos para S. A máquina continua sendo uma máquina com as restrições de BPP pois a proporção de ramos de aceitação e rejeição continua a mesma. Cada ramo de aceitação representa uma interpretação de S tal que $(\mathscr{A},S) \models \varphi(S)$. Logo, como a máquina N tem as restrições de BPP então temos que $|\{S|(\mathscr{A},S) \models \varphi(S)\}| \geq \frac{3}{4} \times 2^{|A|^k}$ ou $|\{S|(\mathscr{A},S) \models \varphi(S)\}| < \frac{1}{4} \times 2^{|A|^k}$. E, dessa forma, $Q_{\geq \frac{3}{4}} S\varphi(S) \in BPFO(Q_{\geq \frac{3}{4}})$. Logo, temos que se $N(bin(\mathscr{A}))$ tem $\frac{3}{4}$ de ramos de aceitação então $\mathscr{A} \models Q_{\geq \frac{3}{4}} S\varphi(S)$ e se $N(bin(\mathscr{A}))$ tem $\frac{1}{4}$ ramos de aceitação então $\mathscr{A} \not\models Q_{\geq \frac{3}{4}} S\varphi(S)$. □

Corolário 6.2.1. *Para todo ε tal que $1 > \varepsilon > \frac{1}{2}$ temos que $BPFO(Q_{\geq \frac{3}{4}})$ caracteriza a classe BPP_ε.*

O corolário acima segue diretamente do teorema acima e do Teorema 3.2.6. Na próxima seção vamos mostrar uma caracterização para a classe RP usando a mesma abordagem apresentada.

6.3 Definibilidade com o Quantificador $Most^k_=$

Nesta seção, vamos mostrar alguns resultados de definibilidade do quantificador generalizado $Most^k_=$. As proposições dessa seção são contribuições do nosso trabalho.

A seguir temos duas proposições que mostram que o $Most^k_{r,=}$ pode definir o $Most^k_=$ e que o $\mathscr{R}^k_=$ consegue definir o $Most^k_{r,=}$. Elas duas seguem diretamente dos resultados de definibilidade do quantificador $Most^k_=$. Todos esses quantificadores de segunda ordem foram mostrados no capítulo 2.

Proposição 6.3.1. $\models Most^k_= X\psi \leftrightarrow Most^k_{r,=} X, X(X = X, \psi)$.

A fórmula da direita expressa que a quantidade de interpretações de X que satisfazem ψ é maior ou igual a quantidade que não satisfaz. Isso quer dizer que a quantidade de interpretações de X que satisfazem ψ é maior ou igual a metade de todas as interpretações de X. Isso é justamente o que é expresso pela fórmula da esquerda.

A proposição abaixo segue da definição do quantificador $Most^k_{r,=}$.

Proposição 6.3.2. $\models Most^k_{r,=} X, Y(\psi,\phi) \leftrightarrow \mathscr{R}^k_= X, Y(\psi \wedge \phi, \psi \wedge \neg\phi)$.

A fórmula da direita diz que a quantidade de interpretações de X que satisfazem $\psi \wedge \phi$ é maior ou igual a quantidade de interpretações de Y que satisfazem $\psi \wedge \neg\phi$. Isso é equivalente ao que expressa a fórmula $Most^k_{r,=}X,Y(\psi,\phi)$.

A seguir vamos mostrar que o quantificador existencial de segunda ordem k-ário pode ser definido com o quantificador $\mathscr{R}^k_=$.

Proposição 6.3.3. *O quantificador $\exists^2_k$ é definível com o quantificador $\mathscr{R}^k_=$.*

$$\exists^2_k X\,\psi(X) \leftrightarrow \mathscr{R}^k_= X,X(\psi(X),\forall x\neg X(x)).$$

A fórmula $\forall x\neg X(x)$ utilizada acima só é satisfeita em uma interpretação da variável de segunda ordem X, a interpretação vazia. Logo, temos que a quantidade de interpretações de X que satisfazem $\psi(X)$ é maior ou igual a 1. Isso é claramente equivalente à existência de uma interpretação de X que satisfaça $\psi(X)$.

Usando a fórmula anterior como abreviação para o existencial de segunda ordem, podemos definir o quantificador "para todo" de segunda ordem.

Proposição 6.3.4. *O quantificador $\forall^2_k$ pode ser definido com o quantificador $Most^k_r$.*

$$\forall^2_k X\,\psi(X) \leftrightarrow \neg\exists^2_k X\neg\psi(X)$$

6.4 Caracterização da Classe *RP*

Nesta seção vamos utilizar nossa abordagem da seção anterior para caracterizar a classe *RP* com uma lógica. Não conhecemos na literatura nenhum resultado de caracterização para a classe *RP*. Primeiro, vamos definir a lógica que vai ser utilizada e em seguida vamos mostrar a caracterização.

Definição 6.4.1 (Lógica $RFO(Most_=)$)**.** *Seja a lógica $FO(Most^k_=)$ definida como visto na Definição 2.3.5 e a lógica $FO(Most_=)$ sendo a lógica $FO((Most^k_=)_{k\in\mathbb{N}})$, ou seja, para todo k natural. Vamos definir o conjunto de fórmulas da lógica $RFO(Most_=)$ como sendo*

$$RFO(Most_=) = \{Most_=R\varphi(R) | \varphi(R) \in FO \text{ e para toda estrutura } \mathscr{A} \text{ temos } \mathscr{A} \models Most_=R\varphi(R) \text{ ou } \mathscr{A} \not\models \exists R\varphi(R)\}.$$

Como restringimos que $\varphi(R) \in FO$ então nas fórmulas $Most_=R\varphi(R)$ existe apenas uma aplicação do quantificador de segunda ordem $Most_=$ não sendo permitido o aninhamento do mesmo. A semântica é definida da forma usual para o quantificador $Most_=$:

Definição 6.4.2. *Adicionamos o caso abaixo na definição da semântica da Lógica de Primeira Ordem.*

$$\mathscr{A} \models Most_=R\varphi(R) \text{ se e somente se } |\{R|(\mathscr{A},R) \models \varphi(R)\}| \geq 2^{|A|^{r-1}}, \text{ onde } r \text{ é a aridade do símbolo relacional } R.$$

Ou seja, pelo menos metade das interpretações do símbolo R devem satisfazer $\varphi(R)$. Agora vamos caracterizar a classe RP através da lógica definida acima.

Teorema 6.4.1 ($RFO(Most_{=}) = RP$)**.** *$RFO(Most_{=})$ caracteriza exatamente RP na classe de estruturas ordenadas.*

Demonstração. ($RFO(Most_{=}) \subseteq RP$).
Seja uma fórmula $Most_{=}R\varphi(R) \in RFO(Most_{=})$.
Podemos construir uma máquina não-determinística N que "chuta" os valores da relação R e depois decide se $(\mathscr{A}, R^{\mathscr{A}}) \models \varphi(R)$, onde $\mathscr{A}$ é uma estrutura de entrada. Como $Most_{=}R\varphi(R)$ é uma fórmula de $RFO(Most_{=})$ então temos que para toda estrutura de entrada $\mathscr{A}$, $|\{R|(\mathscr{A},R) \models \varphi(R)\}| \geq 2^{|A|^{r-1}}$ ou $|\{R|(\mathscr{A},R) \models \varphi(R)\}| = 0$, onde r é a aridade de R. Logo, a máquina N é uma máquina com as restrições da classe RP e temos que para toda estrutura $\mathscr{A}$, se $\mathscr{A} \models Most_{=}R\varphi(R)$ então $N(bin(\mathscr{A}))$ tem pelo menos metade dos ramos de aceitação e se $\mathscr{A} \not\models Most_{=}R\varphi(R)$ então $N(bin(\mathscr{A}))$ não tem nenhum ramo de aceitação.

($RP \subseteq RFO(Most_{=})$).
Seja uma máquina N com tempo n^k e com as restrições da classe RP. Dada uma estrutura de entrada $\mathscr{A}$ saber se N tem um ramo de aceitação com entrada $\mathscr{A}$ é um problema em NP. Pelo Teorema 4.1.2 de Fagin, temos que existe uma fórmula da lógica existencial de segunda ordem $\exists S\varphi(S)$ tal que $\mathscr{A} \models \exists S\varphi(S)$ se e somente se $N(bin(\mathscr{A}))$ tem um ramo de aceitação. Como já foi visto, podemos ajustar a máquina N de forma que a quantidade de ramos de N seja igual a quantidade de interpretações distintas de S. A máquina ajustada continua sendo uma máquina de RP pois mantém a proporção de ramos de aceitação e rejeição. Cada ramo de aceitação representa uma interpretação de S tal que $(\mathscr{A}, S) \models \varphi(S)$. Logo, como a máquina N tem as restrições de RP então temos que $|\{S|(\mathscr{A},S) \models \varphi(S)\}| \geq 2^{k-1}$ ou $|\{S|(\mathscr{A},S) \models \varphi(S)\}| = 0$, onde k é a aridade de S. E, dessa forma, $Most_{=}S\varphi(S) \in RFO(Most_{=})$. Logo, temos que se $N(bin(\mathscr{A}))$ tem pelo menos metade dos ramos de aceitação então $\mathscr{A} \models Most_{=}S\varphi(S)$ e se $N(bin(\mathscr{A}))$ tem zero ramos de aceitação então $\mathscr{A} \not\models Most_{=}S\varphi(S)$. □

Corolário 6.4.1. *Para todo ε tal que $1 > \varepsilon > 0$ temos que $RFO(Most_{=})$ caracteriza a classe RP_{ε}.*

O corolário segue do teorema acima e do Teorema 3.2.1.

6.5 Expressando Problemas em RP com $RFO(Most_{=})$

Vimos no capítulo 3 que o problema $CUT_{=1} \in RP$. Para expressar esse problema com a lógica $RFO(Most_{=})$ poderíamos apenas construir a fórmula a partir da máquina proposta para $CUT_{=1}$ no capítulo 3 e do Teorema 6.4.1 e Corolário 6.4.1 acima. Dessa forma, estaríamos codificando a máquina de RP em uma fórmula da lógica $RFO(Most_{=})$. Mas um ponto interessante da Complexidade Descritiva é que nem sempre precisamos recorrer a este processo. Como vimos para os problemas SAT, $CLIQUE$ e $HORNSAT$, podemos expressar problemas das classes de complexidade sem recorrer às suas máquinas de Turing. A seguir, vamos expressar o problema $CUT_{=1}$ em $RFO(Most_{=})$ sem precisar da máquina que definimos.

$$CUT_{=1} = Most_{=}P(\exists xP(x) \wedge \exists y\neg P(y)) \wedge \exists x\exists y(P(x) \wedge \neg P(y) \wedge E(x,y) \wedge \forall z\forall w((P(z) \wedge \neg P(w) \wedge (z \neq x \vee w \neq y)) \rightarrow \neg E(z,w))).$$

A fórmula diz que na maioria das partições P e $\neg P$ temos que existe apenas uma aresta entre as duas partições. A fórmula acima claramente tem o formato das fórmulas da lógica $RFO(Most_{=})$ mas temos que provar que para toda estrutura que serve de entrada para o problema $CUT_{=1}$ essa estrutura satisfaz a fórmula acima ou não satisfaz a fórmula $\exists P(\exists xP(x) \wedge \exists y\neg P(y)) \wedge \exists x\exists y(P(x) \wedge \neg P(y) \wedge E(x,y) \wedge \forall z\forall w((P(z) \wedge \neg P(w) \wedge (z \neq x \vee w \neq y)) \rightarrow \neg E(z,w)))$.

Primeiramente, temos dois casos: Ou a estrutura que representa o grafo de entrada tem um corte com apenas uma aresta ou não tem esse corte. No caso em que a estrutura que representa um grafo não tem corte com apenas uma aresta, qualquer interpretação para a variável relacional P não vai satisfazer a fórmula $(\exists xP(x) \wedge \exists y\neg P(y)) \wedge \exists x\exists y(P(x) \wedge \neg P(y) \wedge E(x,y) \wedge \forall z\forall w((P(z) \wedge \neg P(w) \wedge (z \neq x \vee w \neq y)) \rightarrow \neg E(z,w)))$.

No caso em que a estrutura tem um corte de tamanho 1 suponha que o corte é composto pela aresta $E(a,b)$. P tem 2^n interpretações e em metade temos $P(a)$ e $\neg P(b)$, ou seja, a e b estão em partições distintas. Isso acontece pelo mesmo argumento que usamos para provar que a máquina não-determinística para o $CUT_{=1}$ respeitava as restrições de RP. Em 2^{n-2} eles estão em P e em 2^{n-2} eles estão em $\neg P$. Então na metade eles estão em partições distintas. Logo, temos que a estrutura satisfaz $Most_{=}P(\exists xP(x) \wedge \exists y\neg P(y)) \wedge \exists x\exists y(P(x) \wedge \neg P(y) \wedge E(x,y) \wedge \forall z\forall w((P(z) \wedge \neg P(w) \wedge (z \neq x \vee w \neq y)) \rightarrow \neg E(z,w)))$.

Pelo argumentos acima, a fórmula realmente é uma fórmula de $RFO(Most_{=})$ e a prova disso é atrelada à prova de que a fórmula realmente expressa o problema. A seguir, vamos discutir a dificuldade de expressar problemas de RP com a lógica $RFO(Most_{=})$ usando como exemplo o problema $MINCUT_{\leq K}$ também mencionado na seção 3.2. Na prova de que esse problema pertencia à classe RP utilizamos o fato de que $RP_{\varepsilon} = RP$ para ε entre 0 e 1. Construir uma fórmula em $RFO(Most_{=})$ é complicado pois não é intuitivo a definição de fórmulas que sejam satisfeitas por uma fração ε de todas as interpretações de uma variável relacional. Além disso, para comparar o tamanho do corte com K é necessário definir existencialmente uma função injetiva assim como fizemos com o problema $CLIQUE$ no Exemplo 4.1.4. Usar o existencial de segunda ordem não é problema pois mostramos que pode ser definido pelo quantificador $Most_{=}$ e então está no formato das fórmulas de $RFO(Most_{=})$. O problema seria provar que a fórmula construída realmente pertenceria à lógica $RFO(Most_{=})$.

Para contornar esse problema, poderíamos definir uma lógica com um número infinito de quantificadores generalizados de segunda ordem, um para cada ε. Adaptando a prova do Teorema 6.4.1 e usando a redução de erro das máquinas de RP do Teorema 3.2.1, poderíamos provar que essa nova lógica com infinitos quantificadores generalizados também caracterizaria a classe RP.

Além de definir os infinitos quantificadores, também seria uma ideia interessante obter resultados de definibilidade de outros quantificadores para cada um dos infinitos quantificadores. Assim poderíamos mais facilmente expressar os problemas com as fórmulas dessa lógica. Por exemplo poderíamos utilizar a fórmula a seguir para expressar o problema $3CLIQUE$.

$$\mathscr{R}^k_{\geq\varepsilon}X,X(\psi\wedge\phi,\phi)$$

onde $\phi = \exists x_1,x_2,x_3(\bigwedge_{i\neq j}x_i \neq x_j \bigwedge_i X(x_i))$ e $\psi = \forall x\forall y((x \neq y \wedge X(x) \wedge X(y)) \rightarrow E(x,y)))$. A semântica desse quantificador é que a quantidade de interpretações que satisfazem a primeira fórmula é maior ou igual uma fração ε da quantidade de interpretações que satisfazem a segunda fórmula. ϕ indica que existem pelo menos 3 elementos na relação X que representa os vértices da clique. ψ expressa que os elementos de X formam uma clique. Vamos deixar como trabalho futuro a definição da lógica mencionada no parágrafo anterior e do estudo de definibilidade de outros quantificadores a partir dessa lógica.

6.6 Caracterização da Classe Probabilística Semântica *coRP*

Agora vamos utilizar nossa abordagem de quantificadores generalizados para definir uma lógica para caracterizar a classe probabilística *coRP*. Nenhum resultado de caracterização para a classe *coRP* foi encontrado na literatura. Primeiro, vamos definir a lógica que vai ser utilizada.

Definição 6.6.1 (Lógica $coRFO(Most_=)$)**.** *Seja a lógica $FO(Most^k_=)$ definida como na Definição 6.4.1 e, logo, a lógica $FO(Most_=) = FO((Most^k_=)_{k\in\mathbb{N}})$. Vamos definir o conjunto de fórmulas da lógica $coRFO(Most_=)$ como sendo*

$$coRFO(Most_=) = \{\forall R\varphi(R) | \varphi(R) \in FO \text{ e para toda estrutura } \mathscr{A} \text{ temos } \mathscr{A} \models \forall R\varphi(R) \text{ ou } \mathscr{A} \not\models Most_=R\varphi(R)\}.$$

A semântica das fórmulas da lógica $coRFO(Most_=)$ é definida da forma usual com a semântica do quantificador $\forall$ de segunda ordem:

Definição 6.6.2. *Como vimos na Definição 2.3.5, adicionamos o caso abaixo na definição da semântica da Lógica de Primeira Ordem obtendo a definição da semântica da lógica $coRFO(Most_=)$.*

$\mathscr{A} \models \forall R\varphi(R)$ *se e somente se para toda interpretação $R^{\mathscr{A}}$ de R temos que $(\mathscr{A},R) \models \varphi(R)$.*

Agora vamos mostrar como podemos caracterizar *coRP* com a lógica $coRFO(Most_=)$.

Teorema 6.6.1 ($coRFO(Most_=) = coRP$)**.** *$coRFO(Most_=)$ captura exatamente coRP na classe de estruturas ordenadas.*

Demonstração. ($coRFO(Most_=) \subseteq coRP$).
Seja uma fórmula $\forall R\varphi(R) \in coRFO(Most_=)$. Podemos construir uma máquina não-determinística N que "chuta" os valores da relação R e depois decide se $(\mathscr{A},R) \models \varphi(R)$, onde $\mathscr{A}$ é uma estrutura de entrada. Como $\forall R\varphi(R)$ é uma fórmula de $coRFO(Most_=)$ então temos que para toda estrutura de entrada $\mathscr{A}$, $|\{R|(\mathscr{A},R) \models \varphi(R)\}| = 2^{|A|^r}$ ou $|\{R|(\mathscr{A},R) \models \varphi(R)\}| < 2^{|A|^{r-1}}$, onde r é a aridade de R. Logo, a máquina N é uma máquina com as restrições da classe *coRP* e temos

que para toda estrutura $\mathscr{A}$, se $\mathscr{A} \models \forall R\varphi(R)$ então $N(bin(\mathscr{A}))$ tem todos os ramos de aceitação e se $\mathscr{A} \not\models \forall R\varphi(R)$ então $N(bin(\mathscr{A}))$ tem menos da metade dos ramos de aceitação.

$(coRP \subseteq coRFO(Most_=))$.

Seja uma máquina não-determinística N com tempo n^k e com as restrições da classe *coRP*. Dada uma estrutura de entrada $\mathscr{A}$ saber se N tem um ramo de aceitação com entrada $\mathscr{A}$ é um problema em *NP*. Pelo Teorema 4.1.2 de Fagin, temos que existe uma fórmula da lógica existencial de segunda ordem $\exists S\varphi(S)$ tal que $\mathscr{A} \models \exists S\varphi(S)$ se e somente se $N(bin(\mathscr{A}))$ tem um ramo de aceitação. Já sabemos que sempre podemos ajustar a máquina N de forma que a quantidade de ramos da máquina seja igual a quantidade de interpretações distintas de S. Cada ramo de aceitação representa uma interpretação de S tal que $(\mathscr{A}, S) \models \varphi(S)$. Logo, como a máquina N tem as restrições de *coRP* então temos que $|\{S|(\mathscr{A}, S) \models \varphi(S)\}| = 2^{|A|^r}$ ou $|\{S|(\mathscr{A}, S) \models \varphi(S)\}| < 2^{|A|^{r-1}}$, onde r é a aridade de S. E, dessa forma, $\forall S\varphi(S) \in coRFO(Most_=)$. Logo, temos que se $N(bin(\mathscr{A}))$ tem todos os ramos de aceitação então $\mathscr{A} \models \forall S\varphi(S)$ e se $N(bin(\mathscr{A}))$ tem menos da metade dos ramos de aceitação então $\mathscr{A} \not\models \forall S\varphi(S)$. □

6.7 Caracterização da Classe Probabilística Semântica *ZPP*

Também não encontramos nenhum resultado de caracterização para a classe *ZPP*. Com a nossa abordagem é possível definir uma lógica para caracterizar essa classe. A seguir, vamos definir a lógica que vai ser utilizada.

Definição 6.7.1 (Lógica $ZPFO(Most_=)$)**.** *Seja a lógica $FO(Most_=)$ definida como anteriormente na Definição 6.4.1. Vamos definir o conjunto de fórmulas da lógica $ZPFO(Most_=)$ como sendo*

$$ZPFO(Most_=) = \{Most_= R_1\varphi_1(R_1) \wedge \forall R_2\varphi_2(R_2) | \varphi_i(R_i) \in FO \text{ e para toda estrutura } \mathscr{A} \text{ temos } \mathscr{A} \models Most_= R_1\varphi_1(R_1) \wedge \forall R_2\varphi_2(R_2) \text{ ou } \mathscr{A} \not\models \exists R_1\varphi_1(R_1) \vee Most_= R_2\varphi_2(R_2)\}.$$

Ou seja, nas fórmulas da lógica $ZPFO(Most_=)$ sempre temos uma conjunção de duas outras fórmulas φ_1 e φ_2 em que a primeira tem apenas uma aplicação do quantificador de segunda ordem $Most_=$ e a segunda inicia com o quantificador de segunda ordem $\forall$. A semântica das fórmulas da lógica $ZPFO(Most_=)$ é definida da forma usual com a semântica dos quantificadores $\forall$ e $Most_=$:

Definição 6.7.2. *Na semântica da lógica $ZPFO(Most_=)$, adicionamos o caso abaixo na definição da semântica da Lógica de Primeira Ordem:*

$$\mathscr{A} \models Most_= R_1\varphi_1(R_1) \wedge \forall R_2\varphi_2(R_2) \text{ se e somente se para toda interpretação de } R_2 \text{ temos que } (\mathscr{A}, R_2) \models \varphi_2(R_2) \text{ e } |\{R_1|(\mathscr{A}, R_1) \models \varphi_1(R_1)\}| \geq 2^{|A|^{r-1}}, \text{ onde } r \text{ é a aridade do símbolo relacional } R_1.$$

Agora vamos mostrar como podemos caracterizar *ZPP* com a lógica $ZPFO(Most_=)$.

Teorema 6.7.1 ($ZPFO(Most_=) = ZPP$)**.** *$ZPFO(Most_=)$ caracteriza exatamente ZPP na classe de estruturas ordenadas.*

Demonstração. $ZPFO(Most_{=}) \subseteq ZPP$. Seja uma fórmula $Most_{=}R_1\varphi_1(R_1) \wedge \forall R_2\varphi_2(R_2) \in ZPFO(Most_{=})$. Podemos construir duas máquinas não-determinísticas N_1 e N_2 tal que N_1 "chuta" os valores da relação R_1 e decide se $(\mathscr{A},R_1) \models \varphi_1(R_1)$. A máquina N_2 "chuta" os valores de R_2 e decide se $(\mathscr{A},R_2) \models \varphi_2(R_2)$. A máquina N com as restrições de ZPP é uma máquina que usa as duas máquinas mencionadas N_1 e N_2. Como $Most_{=}R_1\varphi_1(R_1) \wedge \forall R_2\varphi_2(R_2)$ é uma fórmula de $ZPFO(Most_{=})$ então temos que para toda estrutura de entrada $\mathscr{A}$, $|\{R_1|(\mathscr{A},R_1) \models \varphi_1(R_1)\}| \geq 2^{r_1-1}$ e $|\{R_2|(\mathscr{A},R_2) \models \varphi_2(R_2)\}| = 2^{r_2}$ acontecem ou $|\{R_1|(\mathscr{A},R_1) \models \varphi_1(R_1)\}| = 0$ e $|\{R_2|(\mathscr{A},R_2) \models \varphi_2(R_2)\}| < 2^{r_2-1}$, onde r_1 e r_2 são a aridade de R_1 e R_2, respectivamente. Logo, a máquina N é uma máquina com as restrições da classe ZPP.

$ZPP \subseteq ZPFO(Most_{=})$. Seja uma máquina N com tempo n^k formada por duas outras máquinas N_1 e N_2 e com as restrições da classe ZPP. Dada uma estrutura de entrada $\mathscr{A}$ saber se N_1 tem um ramo de aceitação com entrada $\mathscr{A}$ é um problema em NP. O mesmo vale para N_2. Pelo Teorema 4.1.2 de Fagin, temos que existe uma fórmula da lógica existencial de segunda ordem para cada uma das duas máquinas $\exists S_1\varphi(S_1)$ e $\exists S_2\varphi(S_2)$ tal que $\mathscr{A} \models \exists S_i\varphi(S_i)$ se e somente se $N_i(bin(\mathscr{A}))$ tem um ramo de aceitação para $i \in \{1,2\}$. Já sabemos que sempre podemos ajustar as máquinas N_1 e N_2 de forma que a quantidade de ramos da máquina vai ser igual a quantidade de interpretações distintas de S_1 e S_2, respectivamente. Cada ramo de aceitação de N_i representa uma interpretação de S_i tal que $(\mathscr{A},S_i) \models \varphi(S_i)$ para $i \in \{1,2\}$. Logo, como a máquina N tem as restrições de ZPP então temos que para toda interpretação de S_2, $(\mathscr{A},S_2) \models \varphi_2(S_2)$ e $|\{S_1|(\mathscr{A},S_1) \models \varphi_1(S_1)\}| \geq 2^{|A|^{s_1}-1}$ ou $|\{S_1|(\mathscr{A},S_1) \models \varphi_1(S_1)\}| = 0$ e $|\{S_2|(\mathscr{A},S_2) \models \varphi_1(S_2)\}| < 2^{|A|^{s_2}-1}$, onde s_1 e s_2 são as aridades de S_1 e S_2, respectivamente. E, dessa forma, $Most_{=}S_1\varphi_1(S_1) \wedge \forall S_2\varphi_2(S_2) \in ZPFO(Most_{=})$.

Logo, temos que na máquina N com entrada $bin(\mathscr{A})$ se $N_2(bin(\mathscr{A}))$ tem todos os ramos de aceitação e $N_1(bin(\mathscr{A}))$ tem metade dos ramos de aceitação então $\mathscr{A} \models Most_{=}S_1\varphi_1(S_1) \wedge \forall S_2\varphi_2(S_2)$ e se $N_1(bin(\mathscr{A}))$ tem zero ramos de aceitação e $N_2(bin(\mathscr{A}))$ tem menos da metade de aceitação então $\mathscr{A} \not\models Most_{=}S_1\varphi_1(S_1) \wedge \forall S_2\varphi_2(S_2)$. □

6.8 Caracterização da Classe Semântica $NP \cap coNP$

Agora vamos usar a abordagem apresentada para uma classe não probabilística. Como visto no capítulo 2, a classe $NP \cap coNP$ também é uma classe semântica e, por esse motivo, resolvemos usar nossa abordagem. Nenhum resultado de caracterização foi encontrado na literatura para essa classe. A seguir, vamos definir a lógica que vai ser utilizada.

Definição 6.8.1 (Lógica $(N \cap coN)FO(\exists,\forall)$)**.** *Seja a Lógica de Segunda Ordem $SO = FO(\exists,\forall)$, onde $\exists$ e $\forall$ são quantificadores de segunda ordem, definida na seção 2.2. Vamos definir o conjunto de fórmulas da lógica $(N \cap coN)FO(\exists,\forall)$ como sendo*

$$(N \cap coN)FO(\exists,\forall) = \{\exists R_1\varphi_1(R_1) \wedge \forall R_2\varphi_2(R_2) | \varphi_i(R_i) \in FO \text{ com } i \in \{1,2\} \text{ e para toda estrutura } \mathscr{A} \text{ temos } \mathscr{A} \models \exists R_1\varphi_1(R_1) \wedge \forall R_2\varphi_2(R_2) \text{ ou } \mathscr{A} \not\models \exists R_1\varphi_1(R_1) \vee \forall R_2\varphi_2(R_2).$$

Ou seja, nas fórmulas da lógica $(N \cap coN)FO(\exists,\forall)$ sempre temos uma conjunção de duas fórmulas $\exists R_1\varphi_1(R_1)$ e $\forall R_2\varphi_2(R_2)$ em que a primeira tem apenas uma aplicação do

quantificador de segunda ordem $\exists$ e a segunda inicia com o quantificador de segunda ordem $\forall$ seguido de uma fórmula da Lógica de Primeira Ordem. A semântica das fórmulas da lógica $(N \cap coN)FO(\exists,\forall)$ é definida da forma usual com a semântica dos quantificadores $\forall$ e $\exists$:

Definição 6.8.2. *Na semântica da lógica $(N \cap coN)FO(\exists,\forall)$, adicionamos o caso abaixo na definição da semântica da Lógica de Primeira Ordem:*

$\mathscr{A} \models \exists R_1 \varphi_1(R_1) \wedge \forall R_2 \varphi_2(R_2)$ *se e somente se para toda interpretação de R_2 temos que $(\mathscr{A}, R_2) \models \varphi_2(R_2)$ e existe uma interpretação de R_1 tal que $(\mathscr{A}, R_1) \models \varphi_1(R_1)$.*

Agora vamos mostrar como podemos caracterizar $NP \cap coNP$ com a lógica $(N \cap coN)FO(\exists,\forall)$.

Teorema 6.8.1 $((N \cap coN)FO(\exists,\forall) = NP \cap coNP)$**.** $(N \cap coN)FO(\exists,\forall)$ *captura exatamente $NP \cap coNP$ na classe de estruturas ordenadas.*

Demonstração. $((N \cap coN)FO(\exists,\forall) \subseteq NP \cap coNP)$.
Seja uma fórmula $\exists R_1 \varphi_1(R_1) \wedge \forall R_2 \varphi_2(R_2) \in (N \cap coN)FO(\exists,\forall)$. Podemos construir duas máquinas não-determinísticas N_1 e N_2 tal que N_1 "chuta" os valores da relação R_1 e decide se $(\mathscr{A}, R_1) \models \varphi_1(R_1)$. A máquina N_2 "chuta" os valores de R_2 e decide se $(\mathscr{A}, R_2) \models \varphi_2(R_2)$. A máquina N com as restrições de $NP \cap coNP$ é uma máquina que usa as duas máquinas mencionadas N_1 e N_2. Como $\exists R_1 \varphi_1(R_1) \wedge \forall R_2 \varphi_2(R_2)$ é uma fórmula de $(N \cap coN)FO(\exists,\forall)$ então temos que para toda estrutura de entrada $\mathscr{A}$, $|\{R_1 | (\mathscr{A}, R_1) \models \varphi_1(R_1)\}| > 0$ e $|\{R_2 | (\mathscr{A}, R_2) \models \varphi_2(R_2)\}| = 2^{r_2}$ acontecem ou $|\{R_1 | (\mathscr{A}, R_1) \models \varphi_1(R_1)\}| = 0$ e $|\{R_2 | (\mathscr{A}, R_2) \models \varphi_2(R_2)\}| < 2^{r_2}$, onde r_2 é a aridade de R_2. Logo, a máquina N é uma máquina com as restrições da classe $NP \cap coNP$.

$(NP \cap coNP \subseteq (N \cap coN)FO(\exists,\forall))$.
Seja uma máquina N com tempo n^k formada por duas outras máquinas N_1 e N_2 e com as restrições da classe $NP \cap coNP$. Dada uma estrutura de entrada $\mathscr{A}$ saber se N_1 tem um ramo de aceitação com entrada $\mathscr{A}$ é um problema em NP. O mesmo vale para N_2. Pelo teorema de Fagin, temos que existe uma fórmula da lógica existencial de segunda ordem para cada uma das duas máquinas $\exists S_1 \varphi(S_1)$ e $\exists S_2 \varphi(S_2)$ tal que $\mathscr{A} \models \exists S_i \varphi(S_i)$ se e somente se $N_i(bin(\mathscr{A}))$ tem um ramo de aceitação para $i \in \{1,2\}$. Já sabemos que sempre podemos ajustar as máquinas N_1 e N_2 de forma que a quantidade de ramos das máquinas com mesma entrada $bin(\mathscr{A})$ vai ser igual a quantidade de interpretações distintas de S_1 e S_2, respectivamente. Cada ramo de aceitação de N_i representa uma interpretação de S_i tal que $(\mathscr{A}, S_i) \models \varphi(S_i)$ para $i \in \{1,2\}$. Logo, como a máquina N tem as restrições de $NP \cap coNP$ então temos que para toda interpretação de S_2, $(\mathscr{A}, S_2) \models \varphi_2(S_2)$ e existe interpretação de S_1 tal que $(\mathscr{A}, S_1) \models \varphi_1(S_1)$ ou $|\{S_1 | (\mathscr{A}, S_1) \models \varphi_1(S_1)\}| = 0$ e $|\{S_2 | (\mathscr{A}, S_2) \models \varphi_1(S_2)\}| < 2^{|A|^{s_2}}$, onde s_2 é a aridade de S_2. E, dessa forma, $\exists S_1 \varphi_1(S_1) \wedge \forall S_2 \varphi_2(S_2) \in (N \cap coN)FO(\exists,\forall)$.

Logo, temos que na máquina N com entrada $bin(\mathscr{A})$ se $N_2(bin(\mathscr{A}))$ tem todos os ramos de aceitação e $N_1(bin(\mathscr{A}))$ tem algum ramo de aceitação então $\mathscr{A} \models \exists S_1 \varphi_1(S_1) \wedge \forall S_2 \varphi_2(S_2)$ e se $N_1(bin(\mathscr{A}))$ tem zero ramos de aceitação e $N_2(bin(\mathscr{A}))$ não tem todos de aceitação então $\mathscr{A} \not\models \exists S_1 \varphi_1(S_1) \wedge \forall S_2 \varphi_2(S_2)$. □

7 CONCLUSÕES E TRABALHOS FUTUROS

Neste capítulo vamos comentar as conclusões do trabalho e indicar trabalhos futuros. Nosso trabalho consistiu em criar uma abordagem para conceber resultados de caracterização lógica de classes de complexidade probabilísticas de tempo polinomial. Mostramos que essa abordagem pode ser utilizada para caracterização de classes sintáticas e semânticas, probabilísticas ou não.

De uma forma geral, nossa abordagem consistiu em adaptar a prova de $\#FO = \#P$ e utilizar quantificadores generalizados para simular o comportamento da aceitação de máquinas não-determinísticas. No caso das classes semânticas, tivemos que utilizar lógicas com sintaxe indecidível, ou seja, é indecidível saber se uma fórmula pertence a essa lógica. Como queremos simular as máquinas de uma classe através de uma fórmula e vice-versa, as fórmulas das lógicas tem essa mesma propriedade das máquinas dessas classes.

Dessa forma, os resultados do trabalho podem ser vistos como uma redefinição ou espelhamento das classes de complexidade no contexto de lógicas. Uma questão interessante seria verificar se os demais resultados da área de Complexidade Descritiva podem ser vistos da mesma maneira.

Conseguimos uma prova alternativa para a caracterização da classe PP através da lógica $FO(Most)_{qr\leq 1}$. A primeira prova existente da caracterização segue da caracterização da hierarquia de contagem (CH) pela lógica $FO(Most)$ (KONTINEN, 2009). Nessa caracterização são utilizados os resultados de definibilidade do quantificador $Most^k$. A nossa prova alternativa é mais direta e não precisa desses resultados. Apesar disso, os resultados de definibilidade são úteis para expressar problemas da classe PP mais facilmente. Usando a mesma abordagem da prova alternativa da caracterização de PP, conseguimos capturar a classe $\oplus P$ através da lógica $FO(\oplus)_{qr\leq 1}$.

Para a classe PP também conseguimos mostrar que o problema PP-completo $MAJSAT$ é expressível utilizando a lógica que definimos para caracterizá-la e utilizando os resultados de definibilidade do quantificador $Most^k$. Fizemos o mesmo com o problema $\oplus SAT$ que é $\oplus P$-completo. Deixamos como trabalho futuro a utilização das lógicas definidas para expressar outros problemas dessas classes como por exemplo o $THRESHOLDSAT$ que também é PP-completo. Em (SIMON, 1975), foi mostrado que a classe de linguagens threshold é equivalente à classe PP. Podemos verificar como expressar os problemas dessa classe. Outra classe equivalente à PP é a classe de problemas resolvidos com máquinas de Turing quânticas com pós-seleção $PostBQP$. Esse resultado foi mostrado em (AARONSON, 2004).

Aproveitando os dois resultados de expressar problemas completos acima, outro trabalho futuro seria a prova da caracterização dessas duas classes sintáticas PP e $\oplus P$ utilizando o método de expressar um problema completo da classe com a lógica e mostrar que a classe e a lógica são fechadas sob reduções de primeira ordem.

Também conseguimos expressar um problema da classe RP na lógica $RFO(Most_{=})$. Mostramos a dificuldade de expressar problemas de RP nessa lógica pelo fato de que do ponto

de vista computacional podemos executar um algoritmo várias vezes para aumentar a confiabilidade da resposta. Mesmo que seja possível que $RFO(Most_{=}) = RFO(Q_{\geq\varepsilon})$ para todo ε entre 0 e 1 pode ser complicado expressar problemas em alguma dessas lógicas. Uma solução poderia ser adicionar infinitos quantificadores generalizados de segunda ordem, ou seja, um $Q_{\geq\varepsilon}$ para cada ε tal que $0 < \varepsilon < 1$. Isso não é problema pois a lógica $FO(LFP)$ pode ser vista como a Lógica de Primeira Ordem adicionada de infinitos quantificadores generalizados de segunda ordem. Para facilitar ainda mais a definição de problemas nessa lógica temos que mostrar que os quantificadores

$$Q^k_{2,\geq\varepsilon} = \{\langle A,P,S\rangle \,|P,S \subseteq \mathscr{P}(A^k) \text{ e } |P| \geq |S| \times \varepsilon\}$$

para $0 < \varepsilon < 1$ podem ser definidos a partir dos infinitos quantificadores originais. Vamos deixar como trabalho futuro investigar essas ideias e aplicá-las também nas outras classes semânticas.

Com a nossa abordagem, conseguimos uma caracterização alternativa para a classe *BPP*. Acreditamos que com a lógica que definimos $BPFO(Q_{\geq\frac{3}{4}})$ possa ser mais fácil expressar problemas da classe *BPP* pois as fórmulas são formadas por fórmulas de primeira ordem com a aplicação de um quantificador de segunda ordem. No resultado existente a caracterização era feita a partir da lógica $BPIFP + C$. De qualquer forma, é uma alternativa para expressar os problemas de *BPP* de forma diferente. Como trabalho futuro, podemos fazer uma comparação na definição de problemas de *BPP* usando essas duas lógicas.

Também conseguimos caracterizar as classes probabilísticas de tempo polinomial *RP*, *coRP* e *ZPP*. Até onde sabemos, esses resultados não existiam na literatura. Em todos os casos, utilizamos lógicas definidas de forma análoga às definições das máquinas dessas classes. Além disso, conseguimos expressar problemas da classe *RP* utilizando a lógica $RFO(Most_{=})$. Como trabalho futuro, podemos expressar problemas das classes *coRP* e *ZPP* com as lógicas utilizadas para caracterizá-las.

Neste trabalho, para as classes probabilísticas, focamos apenas nas de tempo polinomial. Podemos utilizar uma adaptação da nossa abordagem para caracterizar classes probabilísticas de espaço logarítmico, de tempo logarítmico ou de tempo exponencial. Como por exemplo, falamos na introdução que já existe o seguinte resultado $BPAC^0 = BPFO$ (KONTINEN, 2009). A adaptação da nossa abordagem para esses outros casos de classes probabilísticas fica como trabalho futuro.

Por fim, no apêndice, usando as provas dos resultado de caracterização lógica de classes de complexidade, desenvolvemos uma forma de gerar algoritmos para problemas dada uma especificação lógica do problema. O algoritmo gerado tem a mesma complexidade da classe que a lógica caracteriza. Fizemos uma comparação de complexidade do algoritmo clássico para o problema *P*-completo *HORNSAT* e do nosso gerador de algoritmos. Percebemos que a complexidade não aumentou muito e que a eficiência do algoritmo gerado depende da formalização lógica do problema. A qualidade da formalização do problema depende tanto da estrutura de entrada como do número de quantificadores da especificação lógica. Fica como

trabalho futuro a implementação do criador de algoritmos para problemas da classe P e também a adaptação dessa ideia para as classes probabilísticas que caracterizamos neste trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

AARONSON, S. Quantum Computing, Postselection, and Probabilistic Polynomial-Time. *CoRR*, abs/quant-ph/0412187, 2004. Disponível em: <http://dblp.uni-trier.de/db/journals/corr/corr0412.html>.

ANDERSSON, A. On Second-order Generalized Quantifiers and Finite Structures. *Ann. Pure Appl. Logic*, v. 115, n. 1-3, p. 1–32, 2002. Disponível em: <http://dblp.uni-trier.de/db/journals/apal/apal115.html>.

ARORA, S.; BARAK, B. *Computational Complexity: A Modern Approach*. 1st. ed. New York, NY, USA: Cambridge University Press, 2009. ISBN 0521424267, 9780521424264.

BADIA, A. *Quantifiers in Action - Generalized Quantification in Query, Logical and Natural Languages*. Kluwer, 2009. 1-156 p. (Advances in Database Systems, v. 37). ISBN 978-0-387-09564-6. Disponível em: <http://dx.doi.org/10.1007/978-0-387-09564-6>.

BARWISE, J.; COOPER, R. Generalized Quantifiers and Natural Language. *Linguistics and Philosophy*, Springer Netherlands, v. 4, n. 2, p. 159–219, jun. 1981. ISSN 0165-0157. Disponível em: <http://dx.doi.org/10.1007/bf00350139>.

COOK, S.; NGUYEN, P. *Logical Foundations of Proof Complexity*. 1st. ed. New York, NY, USA: Cambridge University Press, 2010. ISBN 052151729X, 9780521517294.

COOK, S. A. The complexity of theorem-proving procedures. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM, 1971. (STOC '71), p. 151–158. Disponível em: <http://doi.acm.org/10.1145/800157.805047>.

CORMEN, T. H. et al. *Introduction to Algorithms*. 3rd. ed. [S.l.]: The MIT Press, 2009. ISBN 978-0-262-03384-8.

DOWLING, W. F.; GALLIER, J. H. Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *J. Log. Program.*, v. 1, n. 3, p. 267–284, 1984.

EBBINGHAUS, H.-D.; FLUM, J. *Finite Model Theory*. [S.l.]: Springer, 1995. I-XV, 1-327 p. (Perspectives in Mathematical Logic). ISBN 978-3-540-60149-4.

EBBINGHAUS, H.-D.; FLUM, J.; THOMAS, W. *Mathematical Logic (2. ed.)*. [S.l.]: Springer, 1994. I-X, 1-289 p. (Undergraduate texts in mathematics). ISBN 978-3-540-94258-0.

EICKMEYER, K. *Randomness in Complexity Theory and Logics.* Tese (Doutorado) — Humboldt University of Berlin, 2011. Http://d-nb.info/1015169163. Disponível em: <http://edoc.hu-berlin.de/dissertationen/eickmeyer-kord-2011-08-29/PDF/eickmeyer.pdf>.

EICKMEYER, K.; GROHE, M. Randomisation and Derandomisation in Descriptive Complexity Theory. In: DAWAR, A.; VEITH, H. (Ed.). *CSL*. Springer, 2010. (Lecture Notes in Computer Science, v. 6247), p. 275–289. ISBN 978-3-642-15204-7. Disponível em: <http://dblp.uni-trier.de/db/conf/csl/csl2010.html>.

FAGIN, R. *Contributions to the Model Theory of Finite Structures*. University of California, Berkeley, 1973. Disponível em: <http://books.google.com.br/books?id=dLd7lwEACAAJ>.

GILL, J. Computational Complexity of Probabilistic Turing Machines. *SIAM J. Comput.*, v. 6, n. 4, p. 675–695, 1977.

GRäDEL, E. et al. *Finite Model Theory and Its Applications (Texts in Theoretical Computer Science. An EATCS Series)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN 3540004289.

GUREVICH, Y.; SHELAH, S. Fixed-Point Extensions of First-Order Logic. In: *FOCS*. IEEE Computer Society, 1985. p. 346–353. Disponível em: <http://dblp.uni-trier.de/db/conf/focs/focs85.html>.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN 0321455363.

HUTH, M.; RYAN, M. *Logic in Computer Science: Modelling and Reasoning About Systems*. New York, NY, USA: Cambridge University Press, 2004. ISBN 052154310X.

IMMERMAN, N. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, v. 17, n. 5, p. 935–938, 1988.

IMMERMAN, N. *Descriptive Complexity*. [S.l.]: Springer, 1999. I-XVI, 1-268 p. (Graduate texts in computer science). ISBN 978-0-387-98600-5.

KONTINEN, J. The Hierarchy Theorem for Second Order Generalized Quantifiers. *J. Symb. Log.*, v. 71, n. 1, p. 188–202, 2006. Disponível em: <http://dblp.uni-trier.de/db/journals/jsyml/jsyml71.html>.

KONTINEN, J. A Logical Characterization of the Counting Hierarchy. *ACM Trans. Comput. Logic*, ACM, New York, NY, USA, v. 10, n. 1, p. 7:1–7:21, jan. 2009. ISSN 1529-3785. Disponível em: <http://doi.acm.org/10.1145/1459010.1459017>.

KONTINEN, J. Definability of second order generalized quantifiers. *Arch. Math. Log.*, v. 49, n. 3, p. 379–398, 2010.

LIBKIN, L. *Elements of Finite Model Theory*. [S.l.]: Springer, 2004. ISBN 3-540-21202-7.

LINDSTRÖM, P. First Order Predicate Logic with Generalized Quantifiers. *Theoria*, v. 32, p. 186–195, 1966.

MITZENMACHER, M.; UPFAL, E. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. New York, NY, USA: Cambridge University Press, 2005. ISBN 0521835402.

MOSTOWSKI, A. On a generalization of quantifiers. *Fundamenta Mathematicae*, v. 44, p. 12–36, 1957.

MOTWANI, R.; RAGHAVAN, P. *Randomized Algorithms*. New York, NY, USA: Cambridge University Press, 1995. ISBN 0-521-47465-5, 9780521474658.

PAPADIMITRIOU, C. M. *Computational Complexity*. Reading, Massachusetts: Addison-Wesley, 1994. ISBN 0201530821.

PETERS, S.; WESTERSTåHL, D. *Quantifiers in Language and Logic.* [S.l.]: Clarendon Press, 2006. I-XIX, 1-528 p. ISBN 978-0-19-929125-0.

RAJARAMAN, A.; ULLMAN, J. D. *Mining of Massive Datasets*. New York, NY, USA: Cambridge University Press, 2011. ISBN 1107015359, 9781107015357.

RUSSELL, S. J.; NORVIG, P. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. [S.l.]: Pearson Education, 2010. I-XVIII, 1-1132 p. ISBN 978-0-13-207148-2.

SALUJA, S.; SUBRAHMANYAM, K. V.; THAKUR, M. N. Descriptive Complexity of #P Functions. *J. Comput. Syst. Sci.*, v. 50, n. 3, p. 493–505, 1995.

SIMON, J. *On Some Central Problems in Computational Complexity.* Tese (Doutorado) — Cornell University, Ithaca, NY, USA, 1975. AAI7518004.

SIPSER, M. *Introduction to the Theory of Computation*. 3rd. ed. [S.l.]: Cengage Learning, 2012. ISBN 113318779X.

SZELEPCSÉNYI, R. The Method of Forced Enumeration for Nondeterministic Automata. *Acta Inf.*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 26, n. 3, p. 279–284, nov. 1988. ISSN 0001-5903. Disponível em: <http://dx.doi.org/10.1007/BF00299636>.

THAKUR, M. N. *Descriptive Complexity of Optimization and Counting Problems*. Tese (Doutorado) — University of California at Santa Cruz, Santa Cruz, CA, USA, 1992.

VALIANT, L. G. The Complexity of Computing the Permanent. *Theor. Comput. Sci.*, v. 8, p. 189–201, 1979.

WAGNER, K. W. The Complexity of Combinatorial Problems with Succinct Input Representation. *Acta Inf.*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 23, n. 3, p. 325–356, jun. 1986. ISSN 0001-5903. Disponível em: <http://dx.doi.org/10.1007/BF00289117>.

APÊNDICE A – CRIAÇÃO AUTOMÁTICA DE ALGORITMOS DE TEMPO POLINOMIAL USANDO COMPLEXIDADE DESCRITIVA

No capítulo 3, apresentamos a área da Complexidade Descritiva. No final do capítulo, nos teoremas 4.1.1 e 4.1.3, nós mostramos que $FO \subseteq L$ e que $FO(LFP) = P$. A prova de $FO \subseteq L$ nos mostra como construir uma máquina de Turing de espaço logarítmico para cada fórmula fixa que decide se uma dada estrutura de entrada satisfaz essa fórmula. Veja que podemos usar a forma sistemática de construir essas máquinas da prova do teorema para construir um algoritmo que por sua vez retorna o algoritmo correspondente à máquina de Turing construída na prova. Usando a mesma ideia para $FO(LFP) = P$, podemos ter uma forma de construir algoritmos automaticamente para qualquer problema em P. Para isso, bastaria passar como entrada uma especificação do problema em formato de fórmula da Lógica de Primeira Ordem com operador de Menor Ponto Fixo. Depois de construir o algoritmo automaticamente, bastaria pensar na entrada dele como uma estrutura ordenada. A especificação do problema em $FO(LFP)$ pode ser feito para qualquer problema em P pois já mostramos que para qualquer problema em P temos uma fórmula de $FO(LFP)$.

Neste Apêndice, vamos mostrar, primeiramente, como usar essa ideia para o caso mais simples, ou seja, $FO \subseteq L$. Depois vamos mostrar um exemplo. Em seguida, vamos fazer o mesmo com $FO(LFP) = P$ e mostrar um exemplo de aplicação para uma problema em P. Vamos deixar como trabalho futuro fazer o mesmo para as caracterizações lógicas de classes probabilísticas realizadas neste trabalho.

O Algoritmo $Algoritmo_{FO_L}$ abaixo recebe uma fórmula em primeira ordem φ e retorna um algoritmo $Algoritmo_{\varphi_L}$.

$Algoritmo_{FO_L}(\varphi)$
$AlgoritmoSaida \leftarrow$"$Algoritmo_{\varphi_L}(\mathscr{A})$"
if $qr(\varphi) = 0$ **then**
    $AlgoritmoSaida \leftarrow AlgoritmoSaida$ + "**Return** $\mathscr{A} \models \varphi$"
    EscreveNoArquivo $AlgoritmoSaida$
end if
if $\varphi = \forall x \psi(x)$ **then**
    $AlgoritmoSaida \leftarrow AlgoritmoSaida$ +
    "**for** $a \in A$ **do**
    if $Algoritmo_{\psi(c)_L}(\mathscr{A} \cup \{c^{\mathscr{I}} = a\})$ = **False then** ▷ Estrutura $\mathscr{A}$ é adicionada de uma nova constante c
    Return False

end if
end for
Return True"
EscreveNoArquivo $Algoritmo Saida$
$Algoritmo_{FO_L}(\psi(c))$ ▷ Chamada Recursiva
end if
if $\varphi = \exists x \psi(x)$ **then**
$AlgoritmoSaida \leftarrow AlgoritmoSaida +$
"**for** $a \in A$ **do**
if $Algoritmo_{\psi(c)_L}(\mathscr{A} \cup \{c^{\mathscr{I}} = a\})$ **= True then**
Return True
end if
end for
Return False"
EscreveNoArquivo $Algoritmo Saida$
$Algoritmo_{FO_L}(\psi(c))$ ▷ Chamada Recursiva
end if

Note que o algoritmo acima cria uma rotina para cada quantificador da sentença de entrada. Depois de criar uma rotina para um quantificador, há uma chamada recursiva para o restante da fórmula trocando a variável do quantificador pela nova constante.

A seguir vamos examinar um exemplo de sentença como entrada para esse algoritmo. Vamos executar o acima com uma sentença que representa a propriedade de um grafo ter algum vértice do qual não saem arestas. A sentença é dada por $\varphi = \exists x \forall y \neg E(x,y)$.

Primeiro o algoritmo cria um nome e especifica a entrada do algoritmo de saída, ou seja, "$(\exists x \forall y\ \neg E(x,y))_L(\mathscr{A})$". A execução do algoritmo para essa entrada entra no terceiro "**if**". Depois de executado esse bloco vamos ter nosso programa de saída definido como no Algoritmo abaixo.

$Algoritmo_{\exists x \forall y \neg E(x,y)_L}(\mathscr{A})$
for a $\in A$ **do**
if $Algoritmo_{\forall y \neg E(c,y)_L}(\mathscr{A} \cup \{c^{\mathscr{I}} =$ a $\})$ **= True then**
Return True
end if
end for
Return False

A primeira rotina do nosso algoritmo está totalmente concluída. Note também que o algoritmo $Algoritmo_{\exists x \forall y \neg E(x,y)_L}$ acima está fazendo uma chamada a outro algoritmo. O algoritmo que está sendo chamado vai ser construído no próximo passo. Agora, a execução vai entrar no segundo "**if**". No final, vamos ter o algoritmo abaixo:

$Algoritmo_{\forall y \neg E(c,y)_L}(\mathscr{A})$
for a $\in A$ **do**
if $Algoritmo_{\neg E(c,d)_L}(\mathscr{A} \cup \{d^{\mathscr{A}} =$ a $\})$ **= False then**

Return False
end if
end for
Return False

Mais uma chamada é realizada mas agora entra no primeiro "**if**". O resultado vai ser o Algoritmo $Algoritmo_{\neg E(c,d)_L}$ abaixo:

$Algoritmo_{\neg E(c,d)_L}(\mathscr{A})$
Return $\mathscr{A} \models \neg E(c,d)$

Logo, temos nosso algoritmo que é composto por 3 rotinas. Veja que na última rotina é trivial verificar se $\mathscr{A} \models \neg E(c,d)$. Basta percorrer a estrutura e verificar se a interpretação de c e d pertencem ou não ao predicado E.

Agora podemos dar uma estrutura como entrada e analisar se ela tem a propriedade que desejamos verificar. Vamos supor que a estrutura de entrada seja $\mathscr{G} = \langle\{1,2,3\}, E^{\mathscr{I}} = \{(2,3),(3,2)\}\rangle$. Se, no algoritmo $(\exists x \forall y \neg E(x,y))_L$, o elemento do domínio 1 for atribuído à constante nova c então não importa o valor que seja associado à constante d nova no algoritmo $(\forall y \neg E(c,y))_L$. Qualquer valor atribuído a d na segunda rotina vai resultar que $\mathscr{G} \models \neg E(c,d)$. Logo, o algoritmo $(\exists x \forall y \neg E(x,y))_L$ vai retornar o valor "**True**" que é o esperado.

Agora vamos usar o resultado de que $P = FO(LFP)$ para desenvolver algoritmos que geram algoritmos de problemas em P dada uma especificação lógica do problema. De fato, o algoritmo que vai ser apresentado é baseado na prova de que $FO(LFP) \subseteq P$ mostrado no Teorema 4.1.3. A volta do teorema ($P \subseteq FO(LFP)$) nos garante que todo problema em P tem uma especificação lógica. O algoritmo que vai ser definido vai usar o Algoritmo 1 anterior pois a Lógica $FO(LFP)$ envolve a Lógica de Primeira Ordem e, nesse caso, vamos utilizar o algoritmo anterior que, pelo que vimos, tem como saída um algoritmo de tempo polinomial.

Depois de mostrar o algoritmo, vamos mostrar um exemplo de aplicação para o problema $HORNSAT$. Em seguida, vamos comparar o algoritmo obtido com um algoritmo clássico para o problema que usa tempo linear na quantidade de ocorrências de variáveis proposicionais na fórmula de entrada (DOWLING; GALLIER, 1984; HUTH; RYAN, 2004). O algoritmo de construção de algoritmos de problemas em P é definido abaixo. A entrada do algoritmo é a relação definida indutivamente $LFP_{R_{\langle x1,\ldots,x_k\rangle}}\varphi$ e a fórmula ψ que usa a nova relação.

$Algoritmo_{FO(LFP)_P}(LFP_{R_{\langle x1,\ldots,x_k\rangle}}\varphi, \psi)$
$AlgoritmoSaida \leftarrow$ "$Algoritmo_{\psi_P}(\mathscr{A})$"
$k \leftarrow$ aridade de R
$AlgoritmoSaida \leftarrow AlgoritmoSaida$ + "$R \leftarrow \emptyset$
while R mudar"
for i de 1 até k **do**
$AlgoritmoSaida \leftarrow AlgoritmoSaida$ + "**for** $a_i \in A$"
end for
$AlgoritmoSaida \leftarrow AlgoritmoSaida$ "**if** $Algoritmo\varphi_L(\mathscr{A} \cup R\ c_1^{\mathscr{A}} = a_1, \ldots, c_k^{\mathscr{A}} = a_k)$ = **True**
$R \leftarrow R \cup \{\langle a_1, \ldots, a_k\rangle$ **end if**"

for i de 1 até k **do**
$AlgoritmoSaida \leftarrow AlgoritmoSaida$ "**end for**"
end for
$AlgoritmoSaida \leftarrow AlgoritmoSaida$ + "$\mathscr{A} \leftarrow \mathscr{A} \cup R$
end while
$Algoritmo_{\psi_L}(\mathscr{A})$"
EscreveNoArquivo $AlgoritmoSaida$
$Algoritmo_{FO_L}(\varphi)$
$Algoritmo_{FO_L}(\psi)$

Como exemplo de aplicação do algoritmo acima, vamos utilizá-lo usando como entrada a fórmula

$$HORNSAT = \forall c \exists x((P(c,x) \wedge LFP\varphi(x)) \vee (N(c,x) \wedge \neg LFP\varphi(x))),$$

onde $\varphi(R,x) = \exists c(P(c,x) \wedge \forall y(N(c,y) \rightarrow R(y)))$, vista na prova do Teorema 4.1.3.

Aplicando o algoritmo $FO(LFP)_P$ com entrada $LFP\varphi(R,x)$ e $HORNSAT$ obtemos o algoritmo $HORNSAT_P(\mathscr{A})$ abaixo e suas subrotinas.

$Algoritmo_{HORNSAT_P}(\mathscr{A})$
$R \leftarrow \emptyset$
while R mudar **do**
  for $a_1 \in A$ **do**
    if $Algoritmo_{\varphi(c)_L}(\mathscr{A} \cup R\ c^{\mathscr{A}} = a_1)$ **= True then**
      $R \leftarrow R \cup \{a_i\}$
    end if
  end for
end while
$\mathscr{A} \leftarrow \mathscr{A} \cup R$
$HORNSAT_L(\mathscr{A})$

A complexidade do algoritmo acima é quadrática na quantidade de cláusulas mais a quantidade de variáveis proposicionais. Veja que essa complexidade depende também de como codificamos as fórmulas proposicionais de entrada em estruturas.

O algoritmo resultante criado a partir da fórmula $HORNSAT$ tem complexidade maior que o melhor algoritmo conhecido para o problema da satisfatibilidade de fórmulas em cláusulas de Horn. Apesar disso, a diferença não foi tão grande pois enquanto o melhor é linear na quantidade de ocorrências proposicionais, o que foi construído pelo nosso algoritmo é quadrático na quantidade de cláusulas mais a quantidade de proposições. Fica a pergunta se é possível de expressar esse problema com uma fórmula da Lógica $FO(LFP)$ de forma que a complexidade do algoritmo resultante do nosso algoritmo seja equivalente à complexidade do melhor algoritmo conhecido. No futuro, pretendemos fazer comparações para outros problemas da classe P.

A ideia deste apêndice é interessante de ser usada quando temos que modelar um

situação como uma estrutura e verificar se essa estrutura tem uma certa propriedade formalizada como uma sentença. A criação de algoritmos a partir de uma fórmula que define a propriedade que queremos verificar é útil quando queremos ajustar a estrutura para garantir que certas propriedades sejam satisfeitas. Também é útil quando queremos representar um problema P de uma classe de problemas de decisão C como uma sentença φ e depois obter automaticamente o programa para esse problema apenas passando como entrada para nosso algoritmo a sentença φ.

Outra possibilidade de uso seria quando temos um problema para o qual não conhecemos nenhum algoritmo. Poderíamos definí-lo, se possível, como uma fórmula de $FO(LFP)$ e usar nosso algoritmo para construir um algoritmo para o problema. Outra contribuição interessante do nosso algoritmo é a de difundir os resultados de Complexidade Descritiva para pessoas que não tem conhecimento das áreas mais teóricas da Computação.