Satisfatibilidade de uma fórmula na Lógica Proposicional

Vítor Matos Guimarães
222034360

17 de julho de 2025

1 Introdução

De acordo com Huth e Ryan (HUTH; RYAN, 2004), o objetivo da lógica na ciência da computação é desenvolver linguagens para modelar situações de modo que se possa raciocinar sobre elas formalmente. Podemos representar situações e problemas por meio de, como visto em (NALON, 2025), fórmulas, compostas por símbolos lógicos, que podem ser: símbolos proposicionais, que podem ser verdadeiros ou falsos, operadores, que podem ser unários ou binários e fazem as operações lógicas sobre os símbolos proposicionais, e os símbolos de pontuação, que são responsáveis por delimitar as fórmulas. A seguir está o conjunto de símbolos lógicos da lógica proposicional que serão utilizados nesse trabalho:

1. P = {p,q,r,.., p1, q1, r1,...}, um conjunto enumerável de símbolos;
2. {~, | , &, ->, <->};
3. {(,)}

Para uma fórmula ser considerada bem formada, ou seja, correta, ela deve seguir as seguintes definições, de acordo com (NALON, 2025):

1. Se for um símbolo proposicional sozinho, é uma fórmula bem formada
2. Se for a negação de uma fórmula bem formada, é uma fórmula bem formada
3. Se for uma operação binária de duas fórmulas bem formadas, é uma fórmula bem formada

As fórmulas bem formadas sempre vão representar o resultado de aplicar as operações lógica sobre os símbolos proposicionais, determinando uma valoração a cada símbolo, podendo ser Verdadeiro ou Falso. Como é dito nas notas de aula de (NALON, 2025)

um modelo é um conjunto de valorações que fazem com que a fórmula seja verdadeira como um todo, após aplicar todas as operações lógicas. Para determinar a valoração de uma fórmula φ, devem seguir as seguintes regras, como pode ser visto em (NALON, 2025):

1. O valor de uma fórmula atômica ($p, q, \ldots$) é simplesmente o valor que foi inicialmente atribuído a ela.
2. O valor de ~φ (a negação de φ) será verdadeiro se, e somente se, o valor de φ for falso.
3. O valor de $(\varphi \ \& \ \psi)$ (a conjunção) será verdadeiro se, e somente se, tanto φ quanto ψ forem verdadeiros.
4. O valor de $(\varphi \mid \psi)$ (a disjunção) será verdadeiro se, e somente se, pelo menos um dos dois, φ ou ψ, for verdadeiro.
5. O valor de $(\varphi$ -> $\psi)$ (a implicação) será verdadeiro se, e somente se, o antecedente φ for falso ou o consequente ψ for verdadeiro.
6. O valor de $(\varphi$ <-> $\psi)$ (a bi-implicação) será verdadeiro se, e somente se, φ e ψ tiverem a mesma valoração.

Um dos problemas mais antigos da Lógica Proposicional é a possibilidade de encontrar para uma fórmula qualquer um modelo, ou seja, descobrir uma valoração para cada variável que deixe a fórmula como um todo sendo verdadeira. Esse é um tema de extrema importância para a ciência da computação, como pode ser visto em (COOK, 1971), que classifica o problema da satisfatibilidade booleana como o primeiro problema NP-Completo, ou seja, que no pior caso não pode ser executado em tempo polinomial.

No trabalho que será descrito neste relatório foi criado um programa que encontra um modelo para fórmulas bem formadas da lógica proposicional, caso exista.

2 Metodologia

Para resolver o problema da satisfatibilidade em fórmulas da lógica proposicional, foi criado um programa, utilizando a linguagem de programação C e compilado utilizando gcc na versão 14.2 em um sistema Ubuntu 25.04. Ele recebe como entrada um arquivo de texto "data.txt", contendo fórmulas bem formadas da lógica proposicional com no máximo 31 variáveis cada, mas é recomendado que as fórmulas não possuam mais de 25 variáveis para que o tempo de execução seja aceitável, e gera um arquivo de saída "saida.txt"com uma linha correspondente a cada fórmula de entrada. Cada linha do arquivo de saída contém, primeiramente, "SIM"ou "NÃO", de acordo com a satisfatibilidade da fórmula, seguido de uma lista de pares, delimitada por "[ ]"que é vazia caso a fórmula não seja satisfatível mas, caso contrário, contém as variáveis e a sua respectiva valoração que satisfaça a fórmula.

Essencialmente, o programa foi dividido em 2 estruturas principais: uma que, de maneira análoga a um "Parser", transforma a linha do arquivo de entrada em uma estrutura "Linha", que é composta por outras estruturas chamadas de "Formula", que funciona de maneira similar a uma árvore sintática, e uma lista com as variáveis da fórmula, e outra que verifica todas as possíveis valorações de cada uma das variáveis até encontrar uma que valide a fórmula.

2.1 Parser

O parser é uma estrutura que recursivamente cria uma árvore sintática para a fórmula e uma lista com todas as variáveis. Ele é composto por diversas funções, mas as principais são "parser_init", que inicializa a recursão e a lista de variáveis, e "parser_recursivo", que navega pela linha, caractere por caractere e que recursivamente chama a si mesma.

A tradução da fórmula do arquivo para o programa consiste na criação de uma instância da estrutura "Fórmula", que pode conter até 2 filhos, que são outras fórmulas, e contém um símbolo lógico ou uma variável, caso não tenha mais filhos. Ela procura basicamente 3 tipos de elementos: "(", que é o início de uma fórmula, "~", que indica a negação de uma fórmula, e as variáveis. Sempre que um desses é encontrado, a função se chama novamente para processar aquela subfórmula que foi encontrada.

Quando encontra uma variável, um nó folha, ou seja, um nó terminal apenas com a variável e sem filhos, é criado e a variável é adicionada à lista de variáveis. Para a negação, um nó com apenas um filho é criado e o que está a seguir é processado recursivamente. Ao encontrar parênteses, primeiramente o conteúdo interno é processado, depois o operador binário é identificado e o que está à direita dele é também processado recursivamente, combinando tudo em um nó binário.

O resultado é uma árvore sintática na qual cada nó representa um operador ou variável, e as conexões entre nós respeitam a estrutura da fórmula original. Essa representação em fórmulas e subfórmulas permite, posteriormente, avaliar a fórmula através de um percurso pós-ordem.

2.2 Verificar a satisfatibilidade

Com a fórmula traduzida em forma de árvore sintática, o que falta agora é a verificação da satisfatibilidade. Para isso, o programa utiliza duas funções, a "verifica_sats", que é chamada na função principal do programa, e a função "post_order_valoracao", que navega pela árvore e verifica se a fórmula é satisfeita com a valoração que foi determinada quando foi chamada.

O primeiro passo para a verificação da satisfatibilidade é a contagem do número de possíveis valorações dentro de "verifica_sats". Essa etapa imita o funcionamento de uma tabela verdade, na qual um inteiro sem sinal de 32 bits basicamente reserva um bit para cada uma das variáveis da fórmula que está sendo analisada e testa todos os valores desde 0, no qual todos os bits são 0, logo todas as variáveis são falsas, até um número que pode ser representado como $2^n - 1$, com n sendo o número de variáveis, que faz com que todas

as variáveis tenham o bit que lhes representa como 1, ou seja, verdadeiro.

Para cada possível modelo, representado por 0 até $2^n - 1$, a função chamada de "post_order_valoracao"faz um percurso pós-ordem na árvore e atribui a cada variável a sua respectiva valoração. Desse modo, a valoração de cada filho de cada nó é aplicada no operador daquele nó, até que o operador principal seja utilizado e a fórmula seja determinada como verdadeira ou falsa.

3 Resultados

Para demonstrar o funcionamento do programa, a fórmula $(p \rightarrow (q \wedge r))$ será utilizada como exemplo. Ela foi armazenada no arquivo data.txt, que está no mesmo diretório do arquivo executável do programa, da seguinte forma:

```
(p -> (q & r))
```

Após abrir o arquivo para leitura, o programa lê a linha, caractere por caractere, e cria uma estrutura equivalente à que pode ser vista na Figura 1. Durante a criação, uma lista de variáveis também é criada, armazenando p, q e r.

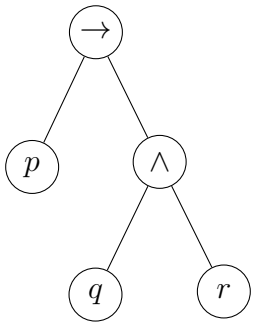

Figura 1: Árvore sintática para a fórmula $(p \rightarrow (q \wedge r))$.

Em seguida, o programa vai testar todas as 8 combinações de valoração, que são representadas por 3 bits em binário como 000, depois 001, depois 010, até chegar em 111, no qual 0 é Falso e 1 é Verdadeiro, ou até encontrar uma valoração que satisfaça a fórmula. Nesse caso, na primeira combinação, 000, ou seja, todas as variáveis com o valor Falso, a fórmula foi satisfeita. No final da execução, o seguinte resultado foi escrito no arquivo de saída:

```
SIM, [[p,F],[q,F],[r,F]]
```

4 Conclusão

No trabalho descrito por meio deste relatório foi criado um programa que consegue com sucesso determinar a satisfatibilidade de uma fórmula da lógica proposicional. A

abordagem escolhida de transformar a linha de texto em árvore sintática e em seguida analisar todas as possíveis combinações de valorações para cada variável garante completude, uma vez que, se a fórmula possuir algum modelo, ele será encontrado.

Infelizmente, o programa tem complexidade exponencial, o que é esperado de um problema NP-Completo, fazendo com que, para um número maior do que o recomendado de 25 variáveis, o tempo de execução seja demasiadamente longo e aumente cada vez mais para cada nova variável introduzida. No entanto, seguindo as recomendações, o programa funciona de acordo com o esperado, com tempo de execução aceitável.

Referências

COOK, S. A. The complexity of theorem-proving procedures. In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*. [S.l.]: ACM, 1971. p. 151–158.

HUTH, M.; RYAN, M. *Logic in Computer Science: Modelling and Reasoning about Systems*. 2. ed. Cambridge: Cambridge University Press, 2004.

NALON, C. *Lógica Computacional 1*. 2025. Notas de aula da disciplina Lógica Computacional 1.