CIC0182: Lógica Computacional 1 - Projeto
Tema: Verificador de Satisfabilidade

Guilherme da Rocha Cunha - 221030007

2025.1

1 Contexto Histórico

Originando-se no período helenístico (32 a.C - 323 a.C) [3], a lógica proposicional é o ramo da lógica que estuda formas de unir e/ou modificar proposições, declarações ou sentenças inteiras para formar proposições, declarações ou sentenças mais complexas, assim como as relações e propriedades lógicas que são derivadas desses métodos de combinar ou alterar declarações [1].

Ela é uma das bases da matemática e hoje é utilizada no projeto de circuitos lógicos em processadores, verificação formal e provas de correção, representação de conhecimento em inteligências artificias e dentre outros.

2 Definições

A seguir encontram-se as definições [2] dos conceitos da lógica proposicional utilizados neste projeto:

2.1 Sintaxe

Definição 1: O conjunto de símbolos lógicos da linguagem proposicional é dada pela união dos seguintes conjuntos:

I. $\mathcal{P} = \{p, q, ..., p_1, q_1, ...\}$: símbolos proposicionais ou variáveis proposicionais;

II. $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$: conectivos lógicos ou operadores lógicos;

III. $\{(,)\}$: símbolos de pontuação.

Definição 2: Uma fórmula é qualquer sequência finita de símbolos lógicos.

Definição 3: A Linguagem Lógica Proposicional $\mathcal{L}_P$ é equivalente ao seu conjunto de fórmulas bem-formadas $\text{FBF}_{\mathcal{L}_P}$, que é definido indutivamente, como se segue:

I. se $p \in \mathcal{P}$, então $p \in \text{FBF}_{\mathcal{L}_P}$;

II. se $\varphi \in \text{FBF}_{\mathcal{L}_P}$, então $\neg\varphi \in \text{FBF}_{\mathcal{L}_P}$;

III. se $\varphi \in \text{FBF}_{\mathcal{L}_P}$ e $\psi \in \text{FBF}_{\mathcal{L}_P}$, então $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$ e $(\varphi \leftrightarrow \psi) \in \text{FBF}_{\mathcal{L}_P}$.

Definição 4: Sejam $\varphi, \psi, \chi \in \text{FBF}_{\mathcal{L}_P}$. Seja $Op : \text{FBF}_{\mathcal{L}_P} \rightarrow \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. O operador principal de φ, $Op(\varphi)$, é dado por:

I. se $\varphi \in \mathcal{P}$, então $Op(\varphi)$ é indefinida;

II. se φ é da forma $\neg\psi$, então $Op(\varphi) = \neg$.

III. se φ é da forma $(\psi * \chi)$ onde $* \in \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$, então $Op(\varphi) = *$.

Definição 5: Sejam $\varphi, \psi, \chi \in \text{FBF}_{\mathcal{L}_P}$. Seja $SubI : \text{FBF}_{\mathcal{L}_P} \rightarrow 2^{\text{FBF}_{\mathcal{L}_P}}$ uma função. O conjunto de subfórmulas imediatas de φ, $SubI(\varphi)$, é dado por:

I. se $\varphi \in \mathcal{P}$, então $SubI(\varphi)$ é indefinido;

II. se φ é da forma $\neg\psi$, então $SubI(\varphi) = \{\psi\}$;

III. se φ é da forma $(\psi * \chi)$ onde $* \in \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$, então $SubI(\varphi) = \{\psi, \chi\}$.

Definição 6: Uma árvore sintática para φ, onde $\varphi \in \text{FBF}_{\mathcal{L}_P}$, é constituída de uma raiz com zero ou mais filhos, dependendo da estrutura de φ:

I. se $\varphi \in \mathcal{P}$, então a raiz é rotulada por φ e tem zero filhos;

II. se φ é da forma $\neg\psi$, então a raiz é rotulada por $\neg$ e tem um único filho, que é a raiz da árvore sintática de ψ;

III. se φ é da forma $(\psi * \chi)$ onde $* \in \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$, então a raiz é rotulada por $*$ e tem dois filhos, onde o da esquerda é a raiz da árvore sintática de ψ e o da direita é a raiz da árvore sintática de χ.

2.2 Semântica

Definição 7: O conjunto $\mathcal{V} = \{V, F\}$ é chamado de conjunto de valores de verdade e cada um de seus elementos é chamado de valor verdade.

Definição 8: Uma valoração booliana $\mathbb{v}_0$ para os símbolos proposicionais de $\mathcal{L}_P$, $\mathcal{P}$, é uma função booliana $\mathbb{v}_0 : \mathcal{P} \rightarrow \mathcal{V}$.

Definição 9: Uma valoração booliana (interpretação) $\mathbb{v}$ para as fórmulas bem-formadas de $\mathcal{L}_P$ é uma função booliana $\mathbb{v} : \text{FBF}_{\mathcal{L}_P} \rightarrow \mathcal{V}$, que estende uma valoração booliana $\mathbb{v}_0 : \mathcal{P} \rightarrow \mathcal{V}$ para símbolos proposicionais $\mathcal{L}_P$, da seguinte forma (onde $\varphi, \psi \in \text{FBF}_{\mathcal{L}_P}$):

I. $\mathbb{v}(\varphi) = \mathbb{v}_0(\varphi)$, se $\varphi \in \mathcal{P}$;

II. $\mathbb{v}(\neg\varphi) = V$ se, e somente se, $\mathbb{v}(\varphi) = F$;

III. $\mathbb{v}(\varphi \wedge \psi) = V$ se, e somente se, $\mathbb{v}(\varphi) = \mathbb{v}(\psi) = V$;

IV. $\mathbb{v}(\varphi \vee \psi) = V$ se, e somente se, $\mathbb{v}(\varphi) = V$ ou $\mathbb{v}(\psi) = V$ ou ambos;

V. $\mathbb{v}(\varphi \to \psi) = V$ se, e somente se, $\mathbb{v}(\varphi) = F$ ou $\mathbb{v}(\psi) = V$ ou ambos;

VI. $\mathbb{v}(\varphi \leftrightarrow \psi) = V$ se, e somente se, $\mathbb{v}(\varphi) = \mathbb{v}(\psi)$.

Definição 10: Seja $\varphi \in \text{FBF}_{\mathcal{L}_P}$. Nós dizemos que φ é satisfatível se existe uma valoração booliana $\mathbb{v} : \text{FBF}_{\mathcal{L}_P} \to \mathcal{V}$ tal que $\mathbb{v}(\varphi) = V$. Neste caso, dizemos que $\mathbb{v}$ é um **modelo** para φ.

3 Algoritmo de decisão

O algoritmo implementado consiste em um verificador de satisfabilidade para fórmulas da lógica proposicional (**Definição 3**), capaz de apresentar um modelo que satisfaça cada fórmula apresentada no arquivo de entrada.

3.1 Árvore sintática

O arquivo `syntax_tree.h` é responsável por definir e representar uma árvore sintática de uma fórmula. Baseando-se na **Definição 4**, **Definição 5** e **Definição 6**, o arquivo é estruturado da seguinte forma:

- `struct Syntax_Tree`: *struct* que representa uma árvore sintática no programa.
- `Syntax_Tree* build_syntax_tree(char* formula)`: função que recebe uma fórmula `formula` e retorna sua árvore sintática.

3.2 Verificador de satisfabilidade

O arquivo `main.c` contém o verificador de satisfabilidade em si. O arquivo contém três principais funcões além da própria função `main`:

- `void count_prop_symb(Syntax_Tree* st, HashTable* ht, List* l)`: função que recebe uma árvore sintática `st` e conta a quantidade de símbolos proposicionais (**Definição 1**) distintos contidos nela. Além de contar, a função também atribui uma numeração i ao i-ésimo símbolo não contabilizado, armazenando em uma tabela hash `ht` e em uma lista `l`.
- `int is_satisfiable(Syntax_Tree* st, HashTable* ht, long long* model)`: função que recebe uma árvore sintática `st` e verifica se a fórmula a qual representa é satisfatível (**Definição 10**).

  A função verifica todas as 2^n valorações boolianas possíveis, onde n é a quantidade símbolos proposicionais distintos na fórmula, por meio de uma *bitmask*. O i-ésimo símbolo proposicional é atribuído o valor verdade V se o i-ésimo *bit* da *bitmask* estiver ligado, ou F caso contrário (**Definição 7** e **Definição 8**).

- `int eval(Syntax_Tree* st, HashTable* ht, long long bool_eval)`: função que recebe um árvore sintática `st` e uma valoração booliana `bool_eval` (**Definição 8**) e retorna um valor verdade (**Definição 9**). Nesse contexto, `bool_eval` é uma *bitmask* de uma valoração booliana qualquer.

3.3 Compilação e Execução

Compilação: Implementado na linguagem C e compilado pelo GCC 14.2.0, o seguinte comando deve ser passado para o terminal:

```
gcc-14 -o sat_checker src/main.c
```

Execução: Para executar o projeto, o seguinte comando deve ser passado para o terminal:

```
./sat_checker <nome do arquivo de entrada>.txt
```

O arquivo de entrada deve ser estruturado de forma que exista apenas uma fórmula bem-formada por linha. Para cada linha do arquivo de entrada:

1. Enquanto não se chegou ao final do arquivo, constroi-se a árvore sintática da fórmula bem-formada presente na linha atual usando a função `build_syntax_tree();`
2. Cria as estruturas de dados necessárias: uma tabela hash e uma lista;
3. Com a função `count_prop_symb()`, conta quantos símbolos proposicionais diferentes existem;
4. Verifica se a fórmula é satisfatível iterando por todas as valorações possíveis por meio da função `is_satisfiable()`. Se for, será impresso `SIM` seguido por uma lista de pares contendo o símbolo proposicional e sua valoração booliana no arquivo de saída. Caso contrário, será impresso `NAO` seguido por uma lista vazia.
5. Vai para a próxima linha do arquivo de entrada e retorna ao passo 1.

3.4 Limitações

O algoritmo de decisão pode apenas avaliar fórmulas com até $1 \leq n \leq 63$ símbolos proposicionais diferentes.

4 Resultados Metateóricos

Corretude: O algoritmo é correto visto que será retornado `SIM` para a fórmula φ se, e somente se, existir um modelo $\mathbb{M}$ tal que $\mathbb{M}$ satisfaz φ.

Completude: O algoritmo é completo visto que se existe um modelo $\mathbb{M}$ tal que $\mathbb{M}$ satisfaz uma fórmula φ, ele retornará `SIM` ou, caso contrário, `NAO`.

Decibilidade: O algoritmo é decidível visto que ele termina em tempo finito com uma resposta `SIM` ou `NAO`.

Complexidade: O algoritmo tem complexidade computacional $\mathcal{O}(2^n)$.

5 Conclusão

Em sua essência, este projeto buscou implementar uma solução de um problema clássico na ciência da computação conhecido como o problema da satisfabilidade booliana, mais conhecido como SAT. Durante o projeto, foi possível notar a dificuldade de se encontrar um algoritmo que encontre uma solução para o problema em um tempo razoável sem utilizar força bruta, tornando claro o motivo no qual o problema pertence à classe de complexidade NP-completo [4].

Referências

[1] KLEMENT, Kevin C.. *Propositional Logic*. Internet Encyclopedia of Philosophy, 2014. Disponível em: https://iep.utm.edu/propositional-logic-sentential-logic/. Acesso em: 01 jul. 2025.

[2] NALON, Cláudia. *Lógica Computacional 1 - Lista de Definições*. Obra não publicada, 2024.2.

[3] BOBZIEN, Susanne. *Ancient Logic*. Stanford Encyclopedia of Philosophy Archive, 2015. Disponível em: https://plato.stanford.edu/archives/spr2016/entries/logic-ancient/. Acesso em: 01 jul. 2025.

[4] KARP, Richard M. *Reducibility among combinatorial problems*. In: MILLER, Raymond E.; THATCHER, James W. (Org.). Complexity of computer computations. New York: Plenum Press, 1972. p. 85-103.