Atividade Prática 2

Rafael Amauri Diniz Augusto - 651047

Questão 1.1:

Expressão = ((A && B) || ^C) || ((A && B)) || C)

c atividade 1.1

p 3 4

-1 -2 0

3 0

-1 -2 0

-3 0

Questão 1.2:

Expressao = (^A && B) || (A && ^B) || (B || A)

Fórmula na CNF

c atividade 1.2

p 2 4

1 -2 0

-1 2 0

-2 0

-1 0

Questão 2: Resolvedor SAT em C

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define MAX_SIZE_CONVERTION_ARRAY 16

// Converte um numero decimal para binario e armazena em um array
short* dec2bin(int num);

// Converte um numero binario (armazenado em array) pra decimal
unsigned long long int bin2dec(short *array);

// Converte um numero binario (armazenado em array com tamanho
explicito) pra decimal.
// Isso é util para arrays que não são do mesmo tamanho que
MAX_SIZE_CONVERTION_ARRAY
unsigned long long int bin2dec_sized_array(short *array, size_t size);

// Faz o trabalho de AC2
int* get_combinations(short *mapa_expressao, size_t
mapa_expressao_size, short *combinacoes);


int main()
{
   clock_t inicio = clock();
   // Abrir o arquivo
   FILE *fp = fopen("dummy.cnf", "r");
   // Linha com buffer de 512 caracteres
   char *linha = malloc(sizeof(char) * 512);
   // Separador
   const char separator[] = " ";
   // Vetor que armazena a linha apos o split
   char *linha_splitted;
   // Armazena numero de variaveis e de clausulas
   int num_vars, num_clausulas;

   // Verificar se o arquivo pode ser acessado
   if (fp == NULL)
```

```
    {
        perror("Erro ao ler o arquivo\n");
        exit(EXIT_FAILURE);
    }


    // Um booleano que indica se o arquivo foi parsed
    bool not_parsed = true;

    // Parsing do arquivo
    while(not_parsed)
    {
        fgets(linha, 500, fp);
        // Ignorar comentarios
        if(linha[0] == 'c')
            continue;

        // Pegar o numero de variaveis e de clausulas
        if(linha[0] == 'p')
        {
            linha_splitted = strtok(linha, separator);
            while(linha_splitted != NULL)
            {
                if(atoi(linha_splitted))
                {
                    num_vars = atoi(linha_splitted);
                    linha_splitted = strtok(NULL, separator);
                    num_clausulas = atoi(linha_splitted);
                }
                linha_splitted = strtok(NULL, separator);
            }

            not_parsed = false;
        }
    }

    printf("N de vars = %d /// N de clausulas = %d\n\n", num_vars,
num_clausulas);

    // Todas as combinacoes possiveis de numeros são 2^num_vars. Aqui é
declarado um vetor com esse numero de combinacoes
    // que é inicializado todo com 0. A ideia para verificar se é SAT é
preencher os resultados das expressões com 1, e se depois
```

```
   // de todas expressões ainda houver pelo menos um 0, é porque é SAT.
Caso não tenha nenhum 0, é UNSAT
   short *combinacoes = (short *) calloc(pow(2, num_vars),
sizeof(short));

   // Declarando um array com num_vars posicoes que vão indicar onde
ficam os valores definidos por cada expressão
   short *mapa_expressao = (short*) malloc((num_vars) * sizeof(short));
   short aux;

   // Setando o array todo para -1
   memset(mapa_expressao, -1, num_vars * sizeof(short));

   not_parsed = true;

   // Iterando todas as expressoes (clausulas)
   for(int i = 0; i < num_clausulas; i++)
   {
       if(fgets(linha, 500, fp) != NULL)
       {
           // Pular comentarios
           if(linha[0] == 'c')
               continue;

           linha_splitted = strtok(linha, separator);

           // Parsing da expressão
           while(not_parsed)
           {
               if(linha_splitted[0] == '0')
               {
                   not_parsed = 0;
                   continue;
               }

               aux = atoi(linha_splitted);

               // Se o numero for negativo na expressao, ele representa
o valor 1 no mapa
               if(aux < 0)
                   mapa_expressao[abs(aux)-1] = 1;
               // Caso nao seja negativo, ele representa o valor 0
               else
```

```
                    mapa_expressao[aux-1] = 0;


                linha_splitted = strtok(NULL, separator);
            }

            not_parsed = 1;

            // Para cada expressao (devidamente armazenada em
mapa_expressao), pegar todas as combinacoes possiveis
            // e armazenar o valor no vetor <combinacoes>
            get_combinations(mapa_expressao, num_vars, combinacoes);

            // Reset do mapa de expressoes para preparar ele pra proxima
expressão
            memset(mapa_expressao, -1, num_vars * sizeof(short));

            printf("Terminei de processar a clausula %d\n\n", i+1);
        }
    }

    bool is_sat = false;

    // Verifica se ainda existe um 0 no vetor de combinacoes apos todas
expressões. Se houver, é SAT. Se não houver, é UNSAT
    for(unsigned long long int i = 0; i < pow(2, num_vars); i++)
    {
        if(combinacoes[i] == 0)
        {
            is_sat = true;
            break;
        }
    }

    printf("\n");

    if(is_sat)
        printf("SAT\n");
    else
        printf("UNSAT\n");


    // Fechar arquivo
```

```
    fclose(fp);

    clock_t fim = clock();
    printf("\nO tempo de execucao do codigo foi de: %lf segundos\n\n",
(double)(fim - inicio) / CLOCKS_PER_SEC);

    return 0;
}


int* get_combinations(short *mapa_expressao, size_t
mapa_expressao_size, short *combinacoes)
{
    // Vai ser usado para contar a quantidade de -1
    short aux = 0;

    // Pegando o numero exato de -1s no mapa da expressão
    for(int i = 0; i < mapa_expressao_size; i++)
    {
        if(mapa_expressao[i] == -1)
            aux++;
    }

    // Criando array pra armazenar onde estão os -1
    short *posicoes_nao_afetadas = (short *)malloc(sizeof(short) * aux);
    aux = 0;

    // Copiando as posicoes dos -1 em array para array_aux
    for(int i = 0; i < mapa_expressao_size; i++)
    {
        if(mapa_expressao[i] == -1)
        {
            posicoes_nao_afetadas[aux] = i;
            aux++;
        }
    }

    // array cheio de 1 que vai ser subtraido
    short array_aux2[aux];

    // populando array_aux2 com 1
    for(int i = 0; i < aux; i++)
    {
```

```
        array_aux2[i] = 1;
    }

    // Para cada combinação possivel, pegar a qual posicao no array de
combinacoes ele corresponde e marcar dita posicao
    // com 1
    for(int i = 0; i <= bin2dec_sized_array(array_aux2, aux); i++)
    {
        short *f = dec2bin(i);

        for(int j = MAX_SIZE_CONVERTION_ARRAY - 1; j >=
MAX_SIZE_CONVERTION_ARRAY - aux; j--)
        {

mapa_expressao[posicoes_nao_afetadas[MAX_SIZE_CONVERTION_ARRAY - j -
1]] = f[j];
        }

        combinacoes[bin2dec_sized_array(mapa_expressao,
mapa_expressao_size)] = 1;

        free(f);
    }
}



short* dec2bin(int num)
{
    short temp_array[MAX_SIZE_CONVERTION_ARRAY], i, aux = 0;
    short *fixed = (short*) calloc(MAX_SIZE_CONVERTION_ARRAY,
sizeof(short));

    for(i = 0; num > 0; i++)
    {
        temp_array[i] = num % 2;
        num = num / 2;
    }

    for(i = i - 1; i >= 0; i--)
    {
        fixed[MAX_SIZE_CONVERTION_ARRAY - 1 - i] = temp_array[i];
        aux++;
    }
```

```
    return fixed;
}


unsigned long long int bin2dec(short *array)
{
    unsigned long long int a = 0;

    for(short i = 0; i < MAX_SIZE_CONVERTION_ARRAY; i++)
        if(array[i] == 1)
            a += pow(2, MAX_SIZE_CONVERTION_ARRAY - 1 - i);


    return a;
}


unsigned long long int bin2dec_sized_array(short *array, size_t size)
{
    unsigned long long int a = 0;

    for(short i = 0; i < size; i++)
        if(array[i] == 1)
            a += pow(2, size - i - 1);


    return a;
}
```

Questão 3.1: Cláusula 3.1

Cláusula H: UNSAT
Cláusula -H: UNSAT

Cláusula I: SAT
Cláusula -I: SAT