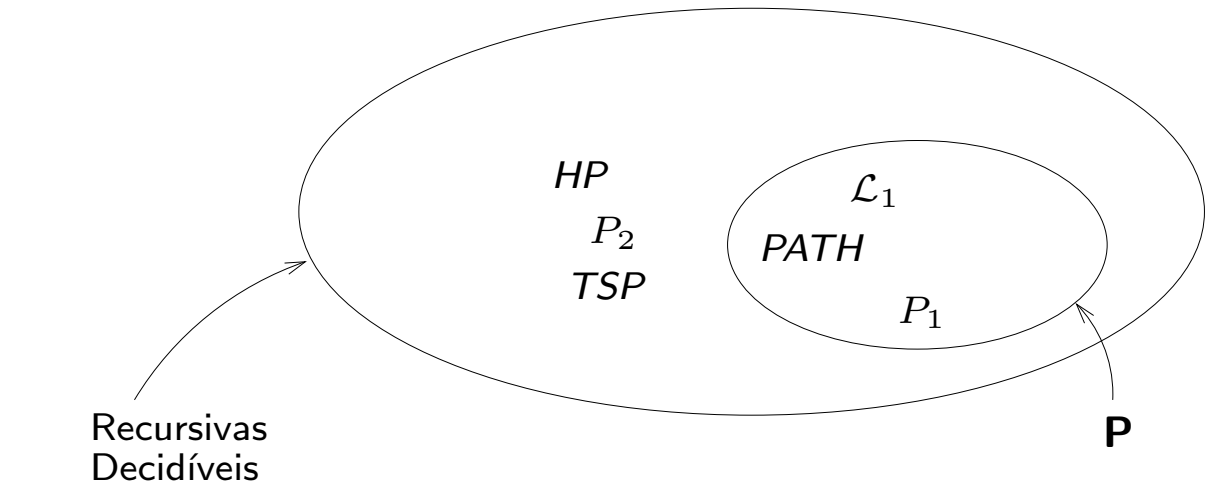
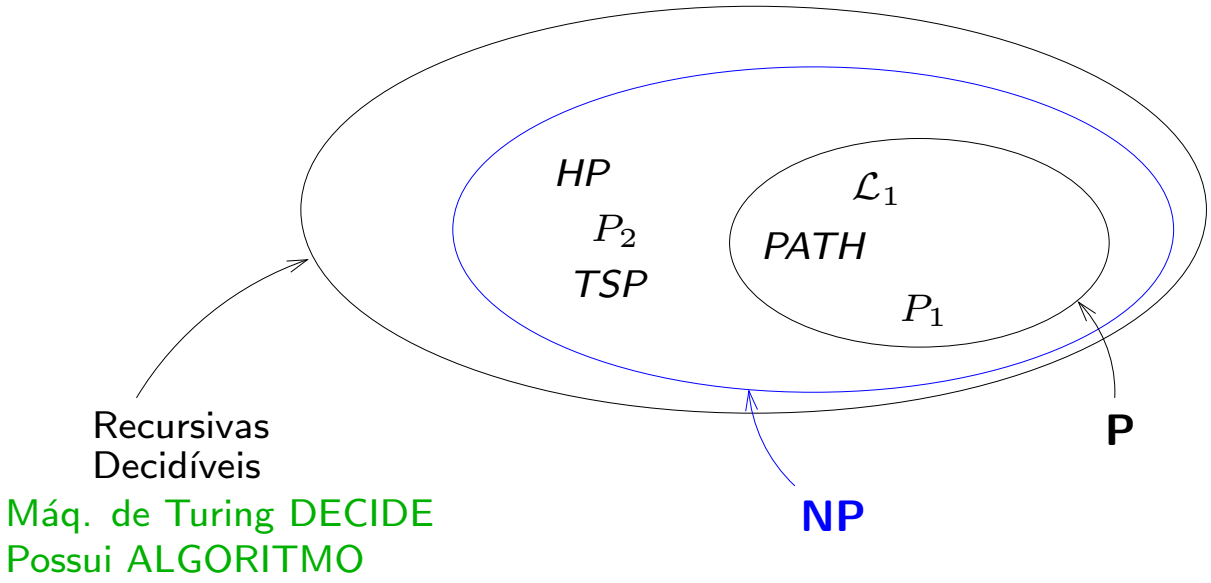
Roteiro da Aula 22

Situação Atual



Máq. de Turing DECIDE
Possui ALGORITMO

Situação Atual



Que classe **NP** é essa?

Por que os americanos estão pagando US$ 1,000,000 para quem provar que **NP**=**P** ou **NP**≠**P**?

A classe **P**

Classe **P**

Um problema A pertence à classe **P** se A possui uma cota superior $O(n^k)$, $k \in \mathbb{N}$. Ou seja, se existe um algoritmo polinomial para A.

Classe **P**

Uma linguagem $\mathcal{L}$ pertence à classe **P** se $\mathcal{L}$ pode ser decidida por uma máquina de Turing determinística de complexidade de tempo de pior caso $O(n^k)$, $k \in \mathbb{N}$.

A classe **NP**

Classe **NP**

Um problema A pertence à classe **NP** se A possui um algoritmo não-determinístico de tempo $O(n^k)$, $k \in \mathbb{N}$.

Classe **NP**

Uma linguagem $\mathcal{L}$ pertence à classe **NP** se $\mathcal{L}$ pode ser decidida por uma máquina de Turing não-determinística de complexidade de tempo de pior caso $O(n^k)$, $k \in \mathbb{N}$.

A classe **NP**

Classe **NP**

Um problema A pertence à classe **NP** se A possui um algoritmo não-determinístico de tempo $O(n^k)$, $k \in \mathbb{N}$.

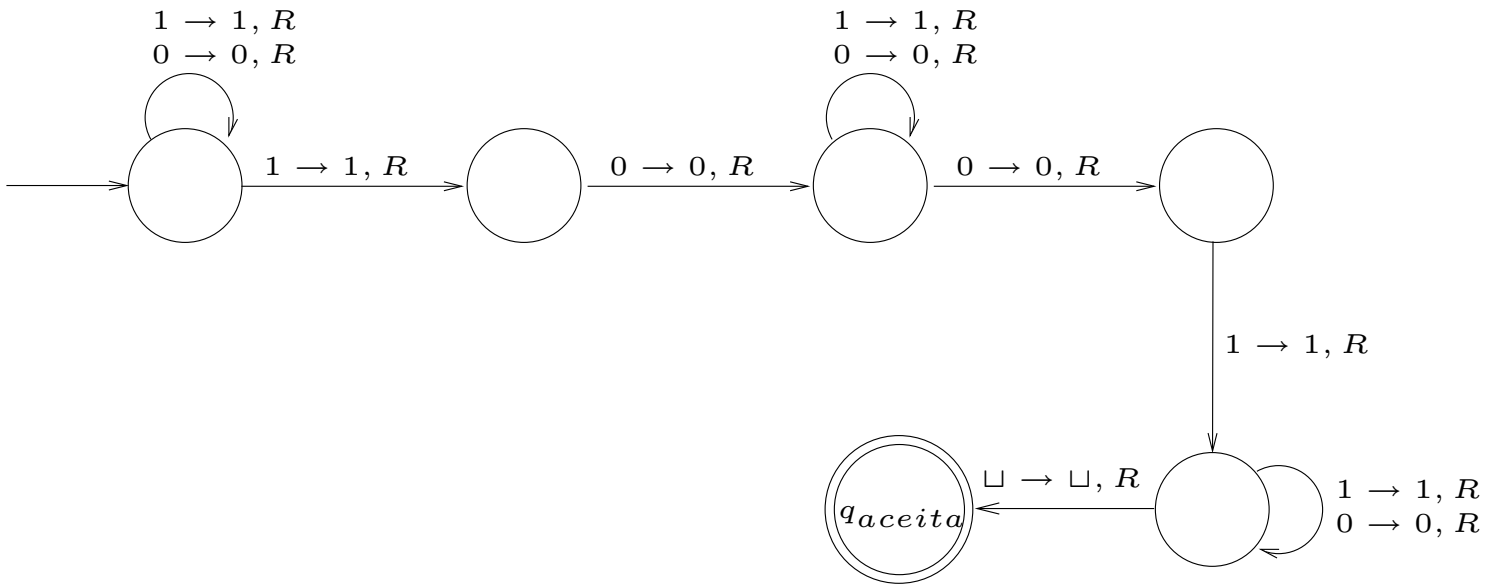
Um problema A pertence à classe **NP** se A pode ser *verificado* por um algoritmo determinístico de tempo $O(n^k)$, $k \in \mathbb{N}$.

Classe **NP**

Uma linguagem $\mathcal{L}$ pertence à classe **NP** se $\mathcal{L}$ pode ser decidida por uma máquina de Turing não-determinística de complexidade de tempo de pior caso $O(n^k)$, $k \in \mathbb{N}$.

Uma linguagem $\mathcal{L}$ pertence à classe **NP** se $\mathcal{L}$ pode ser *verificada* por uma máquina de Turing determinística de complexidade de tempo de pior caso $O(n^k)$, $k \in \mathbb{N}$.

Máq. de Turing Não-determinística



- Determinística (MT): $\delta : Q \times \Sigma \longrightarrow Q \times \Sigma \times \{L, R\}$;
- Não-determinística (NMT): $\delta : Q \times \Sigma \longrightarrow \mathcal{P}(Q \times \Sigma \times \{L, R\})$;

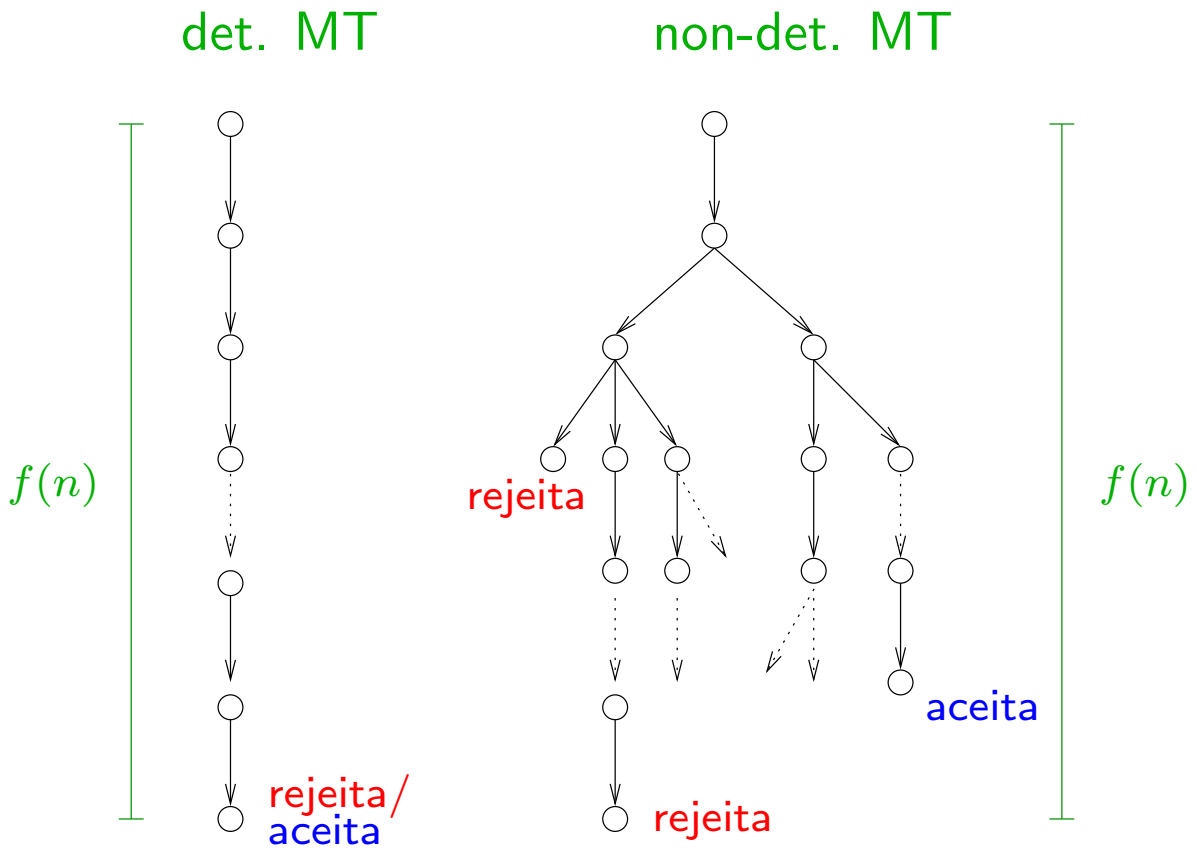
- Semântica: a palavra é aceita se *existe* seqüência de configurações que chega no estado q_{aceita}.

Máq. de Turing Não-determinística

Como já vimos na primeira parte do curso...

- NMT e MT são igualmente expressivas. Quer dizer, tudo o que NMT decide pode ser decidido também por MT;
- NMT não alteram as classes de linguagens/problemas Recursivos/Decidíveis e Recursivamente Enumeráveis.

Máq. de Turing Não-determinística



- Custo de tempo de pior caso: número de configurações da maior seqüência de configurações possível sobre uma dada entrada.

Algoritmo/NMT para P_2

P_2 (SUBSET SUM)

Dado um conjunto C de n números inteiros, positivos ou negativos, decidir se existe um subconjunto de C cuja soma de seus elementos é zero.

```
bool nondetSubsetSum( set C ){
      set S;
1)    S.empty();
2)    for ( int i=0; i < C.length(); i++ ){
2.1)    goto 2.2) OU goto 2);
2.2)    S.append( C[i] );
      }
      int soma = 0;
3)    for ( int j=0; j < S.length(); j++ )
3.1)    soma += S[j];
      if ( soma == 0 ) return true;
      return false;
}
```

Algoritmo/NMT para P_2

- Esse algoritmo pode ser transformado, claramente, numa NMT;
- Para qualquer execução do algoritmo (seqüência de configurações da NMT), o custo será linear $O(n)$;
- O algoritmo retorna true se e somente se existe um subconjunto cuja soma é zero.

Conclusão: P_2 (SUBSET SUM) pertence à **NP**

Algoritmo/NMT para P_2

```
bool nondetSubsetSum( set C ){
      set S;
1)    S.empty();
2)    for ( int i=0; i < C.length(); i++ ){
2.1)    goto 2.2) OU goto 2);
2.2)    S.append( C[i] );
      }
      int soma = 0;
3)    for ( int j=0; j < S.length(); j++ )
3.1)    soma += S[j];
      if ( soma == 0 ) return true;
      return false;
}
```

Verificação determinística

Algoritmo/NMT para P_2

```
bool nondetSubsetSum( set C ){
      set S;
1)    S.empty();
2)    for ( int i=0; i < C.length(); i++ ){
2.1)    goto 2.2) OU goto 2);
2.2)    S.append( C[i] );
      }
      int soma = 0;
3)    for ( int j=0; j < S.length(); j++ )
3.1)    soma += S[j];
      if ( soma == 0 ) return true;
      return false;
}
```

Verificação determinística

- Passos 1) a 2.2): constrói conjunto S não-deterministicamente;

Algoritmo/NMT para P_2

```
bool nondetSubsetSum( set C ){
      set S;
1)    S.empty();
2)    for ( int i=0; i < C.length(); i++ ){
2.1)    goto 2.2) OU goto 2);
2.2)    S.append( C[i] );
      }
      int soma = 0;
3)    for ( int j=0; j < S.length(); j++ )
3.1)    soma += S[j];
      if ( soma == 0 ) return true;
      return false;
}
```

Verificação determinística

- Passos 1) a 2.2): constrói conjunto S não-deterministicamente;
- Passo 3) e 3.1): verifica se S realmente certifica C.

Verificação determinística

Verificação determinística

Um problema A pode ser *verificado* polinomialmente, se é possível, ao responder ao problema, apresentar um certificado que pode ser verificado deterministicamente em tempo de pior caso $O(n^k)$, $k \in \mathbb{N}$.

Formalmente, um Verificador para uma linguagem (problema) $\mathcal{L}$ é uma MT determinística (algoritmo determinístico), A, tal que:

$$\mathcal{L} = \{w \mid \text{existe } c \text{ tal que } A \text{ aceita } \langle w, c \rangle\}$$

Verificação determinística

Classe **NP** (def. original)

Uma linguagem $\mathcal{L}$ pertence à classe **NP** se $\mathcal{L}$ pode ser decidida por uma máquina de Turing não-determinística de complexidade de tempo de pior caso $O(n^k)$, $k \in \mathbb{N}$.

Classe **NP** (def. mais usada)

Uma linguagem $\mathcal{L}$ pertence à classe **NP** se $\mathcal{L}$ pode ser *verificada* por uma máquina de Turing determinística de complexidade de tempo de pior caso $O(n^k)$, $k \in \mathbb{N}$.

Exemplos

- O problema do *Hamiltonian Path* pertence à **NP**? Ou seja, pode ser verificado em tempo polinomial?

Para responder a essa pergunta precisamos de duas coisas:

Exemplos

- O problema do *Hamiltonian Path* pertence à **NP**? Ou seja, pode ser verificado em tempo polinomial?

Para responder a essa pergunta precisamos de duas coisas:

- Qual é o certificado?

- Dá para verificar o certificado em tempo polinomial?

Exemplos

- O problema do *Hamiltonian Path* pertence à **NP**? Ou seja, pode ser verificado em tempo polinomial?

Para responder a essa pergunta precisamos de duas coisas:

- Qual é o certificado?
  - Uma permutação dos vértices: $v_{i_1}, v_{i_2}, \ldots, v_{i_n}$
- Dá para verificar o certificado em tempo polinomial?
  - Sim, claro, consultando a matriz de adjacências...

Exemplos

- Note que existe uma grande *assimetria* entre as respostas `sim` e `não`, no que diz respeito à verificação do HP:
  - Se a resposta é `sim` existe um certificado que pode ser verificado polinomialmente: a permutação de vértices;
  - Mas, e se a resposta é `não`? Qual é o certificado?

Exemplos

- Note que existe uma grande *assimetria* entre as respostas `sim` e `não`, no que diz respeito à verificação do HP:
  - Se a resposta é `sim` existe um certificado que pode ser verificado polinomialmente: a permutação de vértices;
  - Mas, e se a resposta é `não`? Qual é o certificado?

O problema $\overline{\text{HP}}$ não parece pertencer à **NP**...

Exemplos

- O *Traveling Salesman Problem* pertence à **NP**?

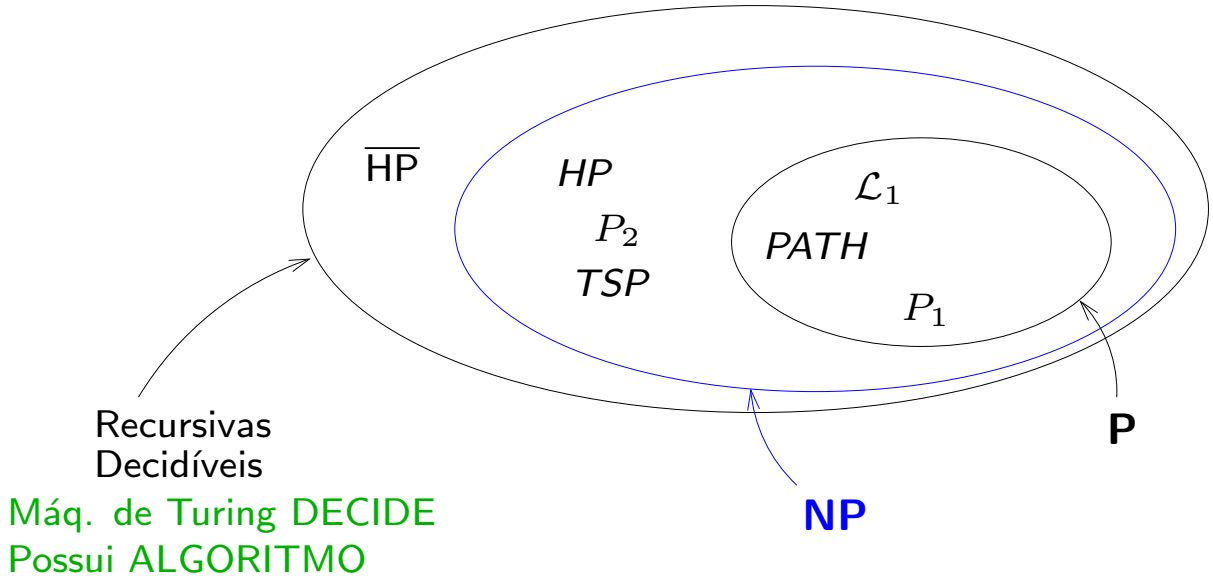
Exemplos

- O *Traveling Salesman Problem* pertence à **NP**?
- Qual é o certificado?
- Dá para verificar o certificado em tempo polinomial?

Exemplos

- O *Traveling Salesman Problem* pertence à **NP**?
- Qual é o certificado?
  - Novamente, uma permutação dos vértices: $v_{i_1}, v_{i_2}, \ldots, v_{i_n}$
- Dá para verificar o certificado em tempo polinomial?
  - Sim, consultando a matriz de adjacências, somando os custos e verificando, ao final, se é menor que s...

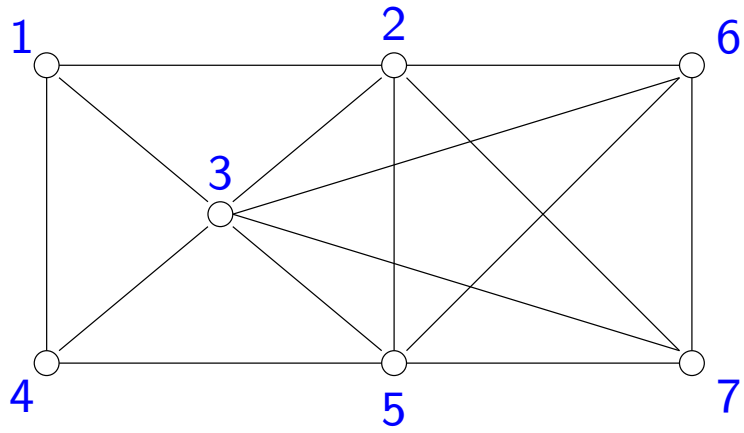
Situação Atual



Vamos ver mais exemplos...

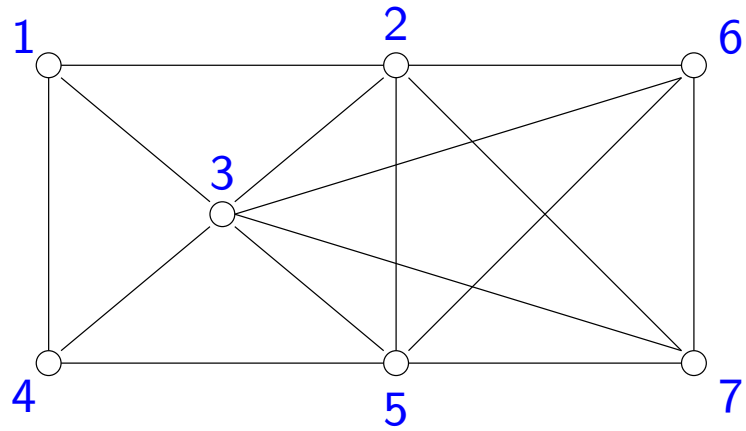
Clique

- Dado um grafo não-direcionado $G = (V, E)$, uma clique em G é um subconjunto de vértices, $V_c \subseteq V$, tal que: se $a \in V_c$ e $b \in V_c$, então $(a, b) \in E$.



Clique

- Dado um grafo não-direcionado $G = (V, E)$, uma clique em G é um subconjunto de vértices, $V_c \subseteq V$, tal que: se $a \in V_c$ e $b \in V_c$, então $(a, b) \in E$.



- Problema CLIQUE:
  - Dado um grafo $G = (V, E)$ e um inteiro positivo k, decidir se existe uma clique de tamanho k em G.

CLIQUE pertence à **NP**? CLIQUE pertence à **P**?

Lógica Proposicional

- Variável Booleana, $x, y, z, \ldots$: assume valores 1 ou 0 (verdadeiro ou falso);
- Operações Booleanas: OU: $x \vee y$, E: $x \wedge y$, NÃO: $\overline{x}$;
- Fórmula Booleana: expressão envolvendo variáveis e operações booleanas. Exemplo:

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

Lógica Proposicional

Semântica: em que resulta uma operação?

Tabelas-verdade

- NÃO: $\overline{0} = 1$, $\overline{1} = 0$;
- OU: $0 \vee 0 = 0$, $0 \vee 1 = 1$, $1 \vee 0 = 1$, $1 \vee 1 = 1$;
- E: $0 \wedge 0 = 0$, $0 \wedge 1 = 0$, $1 \wedge 0 = 0$, $1 \wedge 1 = 1$;

Lógica Proposicional

Satisfatibilidade

Uma fórmula ϕ é satisfatível se existe uma valoração às variáveis de ϕ que a faz resultar em 1 (verdadeiro).

- Exemplo: $\phi_1 = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$:

- Exemplo: $\phi_2 = \overline{(x \vee y)} \wedge z \wedge \overline{(z \wedge \overline{y})}$:

Lógica Proposicional

Satisfatibilidade

Uma fórmula ϕ é satisfatível se existe uma valoração às variáveis de ϕ que a faz resultar em 1 (verdadeiro).

- Exemplo: $\phi_1 = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$:
  - Satisfatível, para $x = 0$, $y = 1$ e $z = 0$;
- Exemplo: $\phi_2 = \overline{(x \vee y)} \wedge z \wedge \overline{(z \wedge \overline{y})}$:
  - Insatisfatível...

Lógica Proposicional

- Literal: é uma variável ou sua negação.

Forma Normal Conjuntiva (CNF)

Uma fórmula ϕ é uma CNF-fórmula se ϕ é uma conjunção de disjunções de literais.

- Ou seja, é da forma: $\phi = \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_k$;
- onde, $\phi_i = \beta_1 \vee \beta_2 \vee \cdots \vee \beta_\ell$;
- onde β_i são literais.

Ex.: $(\underbrace{x_1}_{\text{literal}} \vee \underbrace{\overline{x_2}}_{\text{literal}} \vee x_4 \vee x_5) \wedge (x_3 \vee \overline{x_4}) \wedge \underbrace{(x_3 \vee \overline{x_6} \vee \overline{x_1})}_{\text{cláusula}}$

Lógica Proposicional

k-Forma Normal Conjuntiva (kCNF)

Uma fórmula ϕ é uma kCNF-fórmula se é uma CNF-fórmula onde todas as cláusulas têm k literais.

Temos 3 problemas muito importantes:

- SAT $= \{\phi \mid \phi$ é uma fórmula satisfatível$\}$;
- 3SAT $= \{\phi \mid \phi$ é uma 3CNF-fórmula satisfatível$\}$;
- 2SAT $= \{\phi \mid \phi$ é uma 2CNF-fórmula satisfatível$\}$.

SAT, 3SAT e 2SAT

- SAT pertence à **NP**? SAT pertence à **P**?
- 3SAT pertence à **NP**? 3SAT pertence à **P**?
- 2SAT pertence à **NP**? 2SAT pertence à **P**?