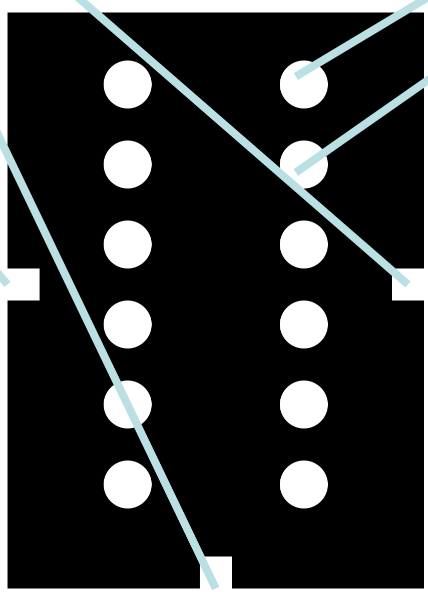
A Classe **NP**

Agenda

- Complexidade Não Determinista
- A classe **NP**
  - Definida por…
    - …aceitação em tempo polinomial por NTM's
    - … instâncias positivas com provas de tamanho polinomial
      …aceitação por verificadores em tempo polinomial
  - Exemplos em:
    - Jogo de cartões perfurados
    - SAT (satisfazibilidade de expressões Booleanas)
      - Variantes: CSAT, *n*SAT
- A classe Co-**NP**
- Reduções de tempo polinomial

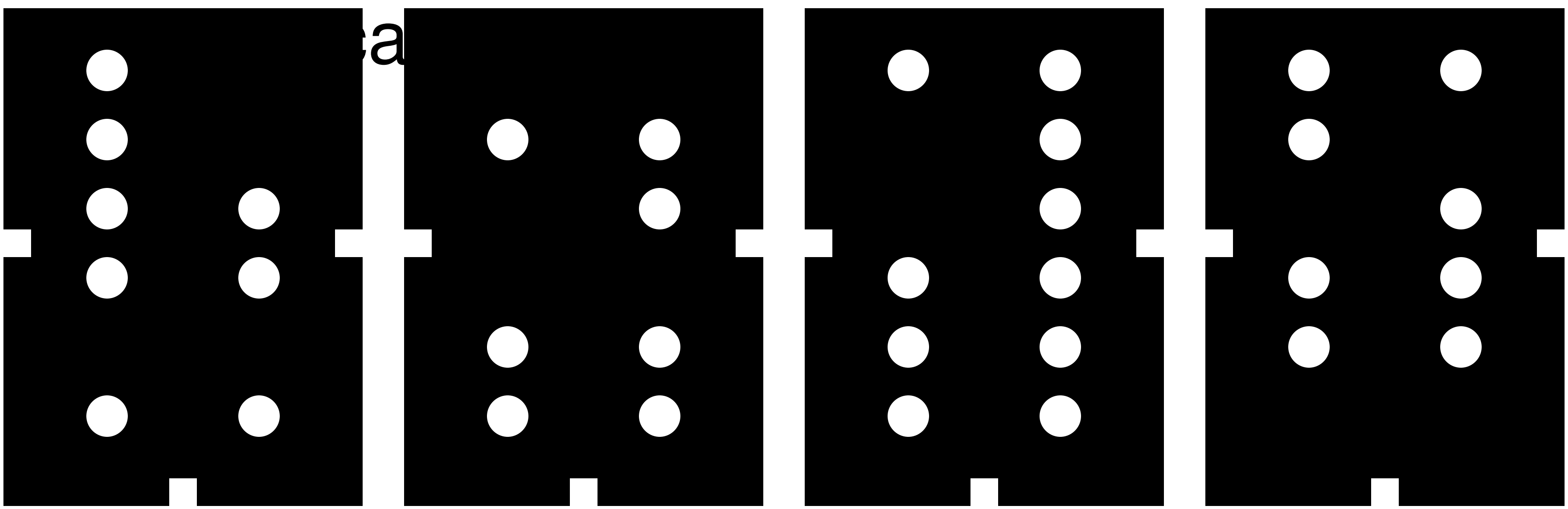
Jogo de cartões perfurados

Considere cartões perfurados que têm **3 fendas** e certo número de **furos** arranjados em duas colunas



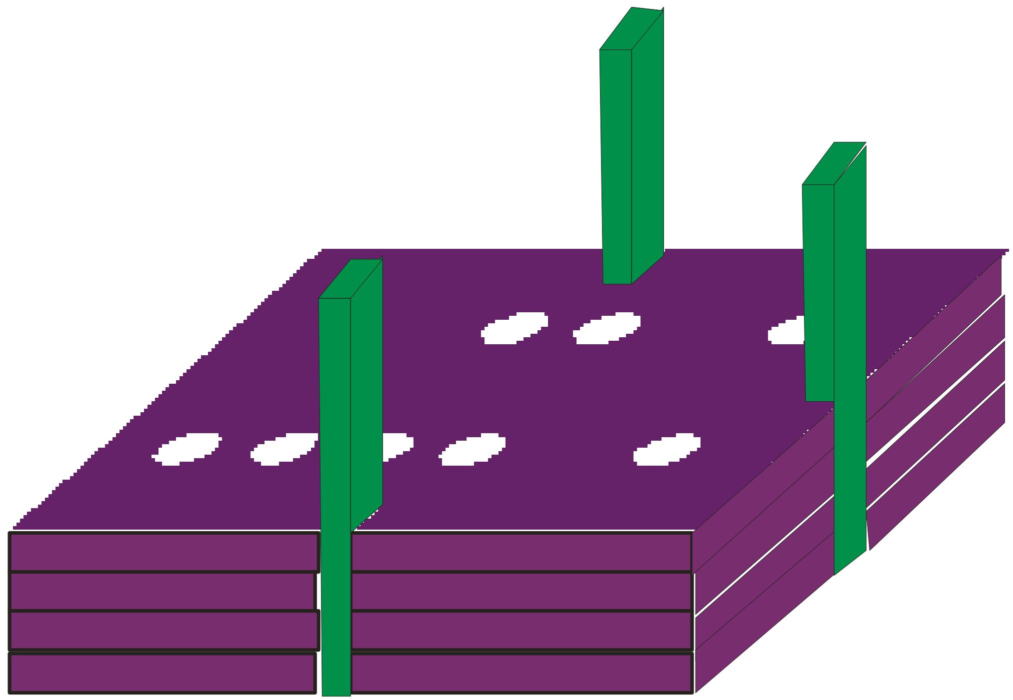
Jogo de cartões perfurados

São sempre 3 fendas, usadas para encaixar os cartões em 3 hastes, mas o número e posição dos furos varia em ca



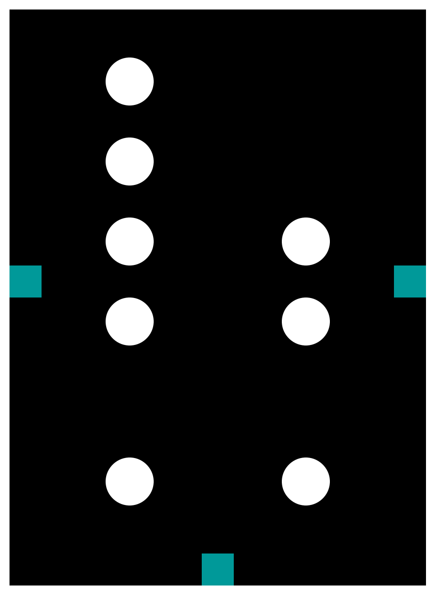
Jogo de cartões perfurados

Veja como os cartões são encaixados nas hastes:



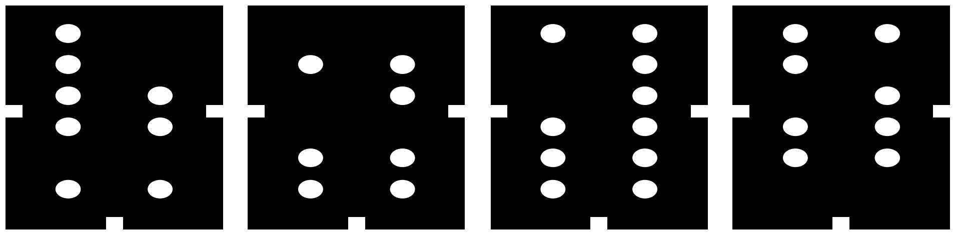
Jogo de cartões perfurados

O objetivo do jogo é colocar todos os cartões nas hastes, de modo que todos os furos sejam tapados. É permitido virar os cartões.



Jogo de cartões perfurados

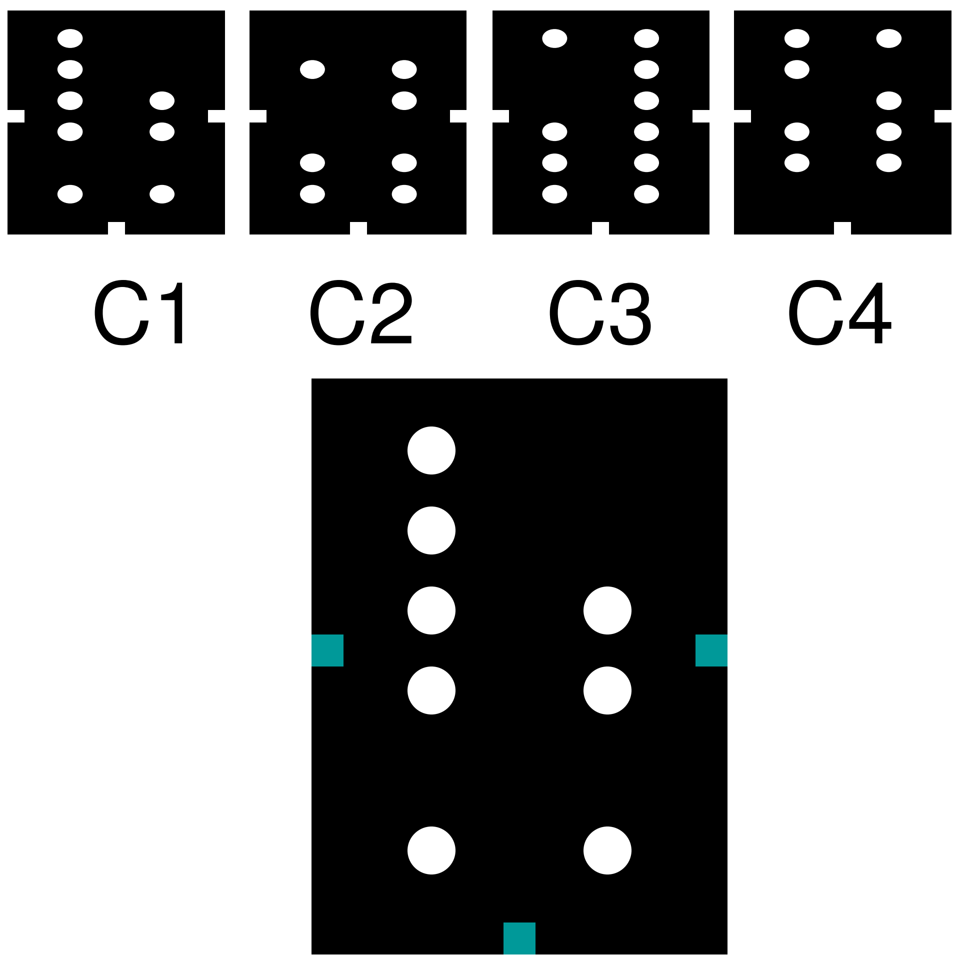
Q: Encontre uma solução p/ a instância:



C1 C2 C3 C4

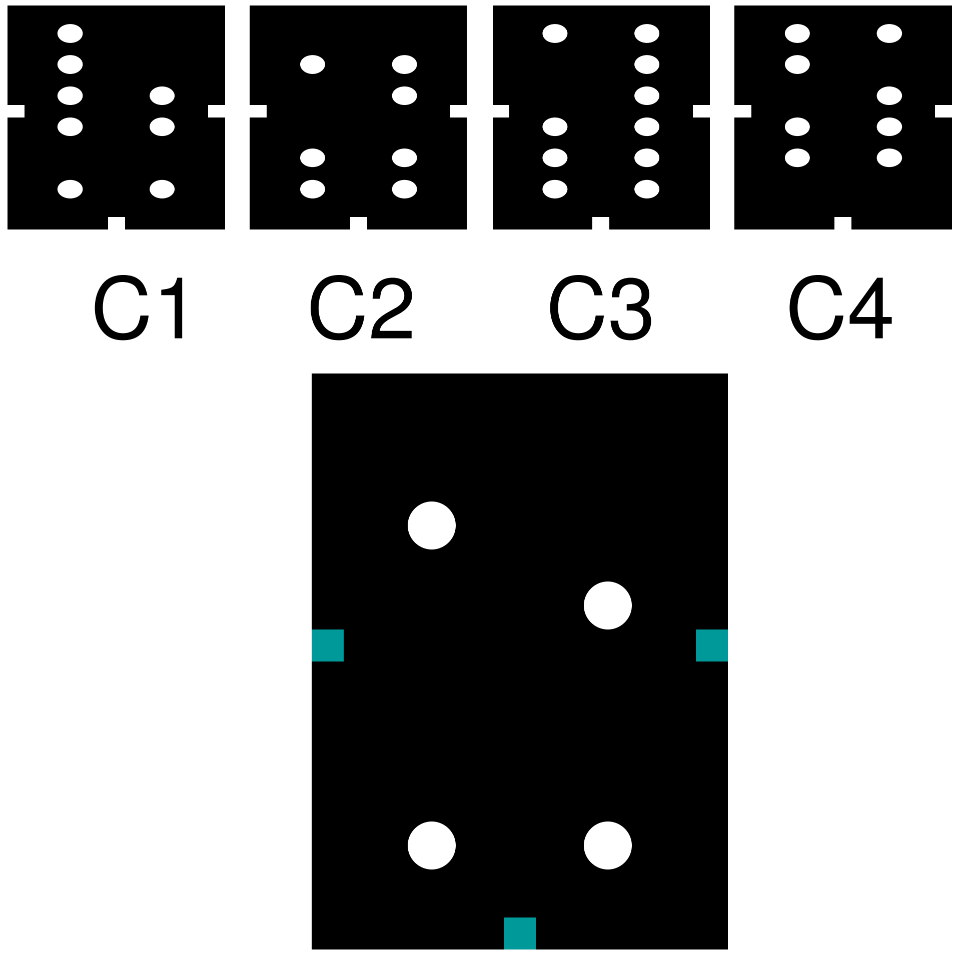
Jogo de cartões perfurados

R: Uma solução é C1, C2, C3, flip(C4):



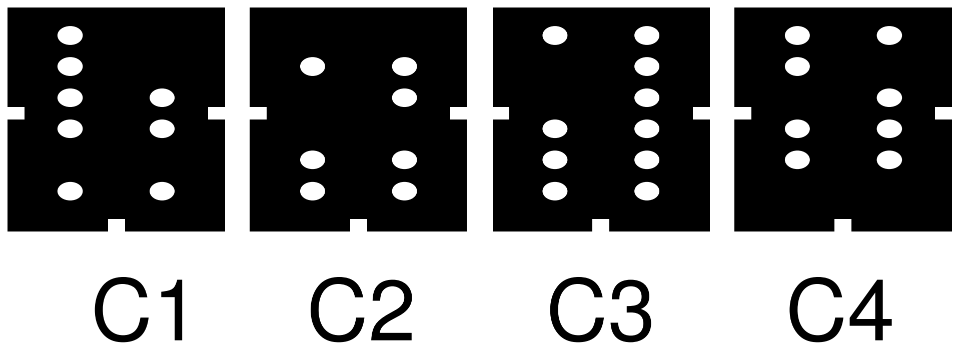
Jogo de cartões perfurados

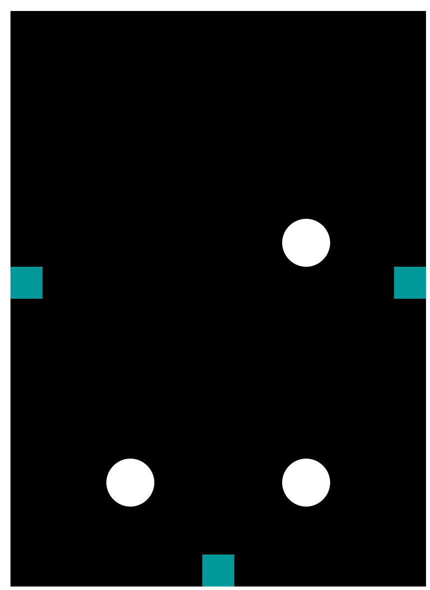
R: Uma solução é C1, C2, C3, flip(C4):



Jogo de cartões perfurados

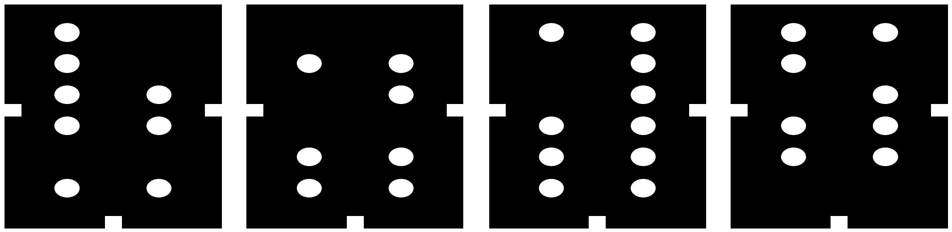
R: Uma solução é C1, C2, C3, flip(C4):



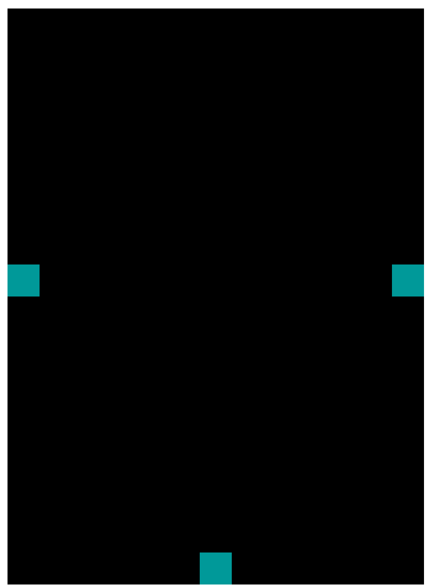


Jogo de cartões perfurados

R: Uma solução é C1, C2, C3, flip(C4):



C1 C2 C3 C4



Jogo de cartões perfurados

Q: O jogo pode ser resolvido de maneira sistemática (por um algoritmo)?

O jogo de cartões perfurados é Decidível

A: Sim. Um algoritmo para a solução seria:

```
SolvePuzzle(cards C1, C2, … Cn)
   for(i1= 0 to 1)
        if (i1= 1) flip C1
        for(i2= 0 to 1)
                if (i2= 1) flip C2
                        …
                        for(in= 0 to 1)
                                if (in= 1) flip Cn
                                put cards on pegs
                                if (no holes) ACCEPT
   REJECT
```

Q: Qual é o tempo de execução?

O jogo de cartões perfurados é Decidível

R: Tempo de execução: No pior caso, rejeição, temos que seguir por todas as possíveis iterações: *n* for-loops aninhados, cada um com duas possibilidades. Então o tempo de execução no pior caso é $\Omega(2^n)$. Portanto, o algoritmo tem tempo de execução exponencial.

Q: Você poderia definir um algoritmo de tempo de execução polinomial?

O jogo de cartões perfurados é NP-completo

A: Se puder, você ganha $1,000,000!!!

Isso porque esse problema é NP-completo, como vamos explicar nas próximas aulas. Prover uma solução eficiente para esse problema implica obter uma solução eficiente para **milhares** de outros problemas algoritmicos importantes em computação, incluindo a quebra do sistema criptográfico RSA!

Tempo de Execução Não Determinista

Lembre-se que em uma TM não determinista, uma computação consiste em uma árvore de computação, cujos ramos, da raiz para as folhas, são as várias computações que poderiam ocorrer.

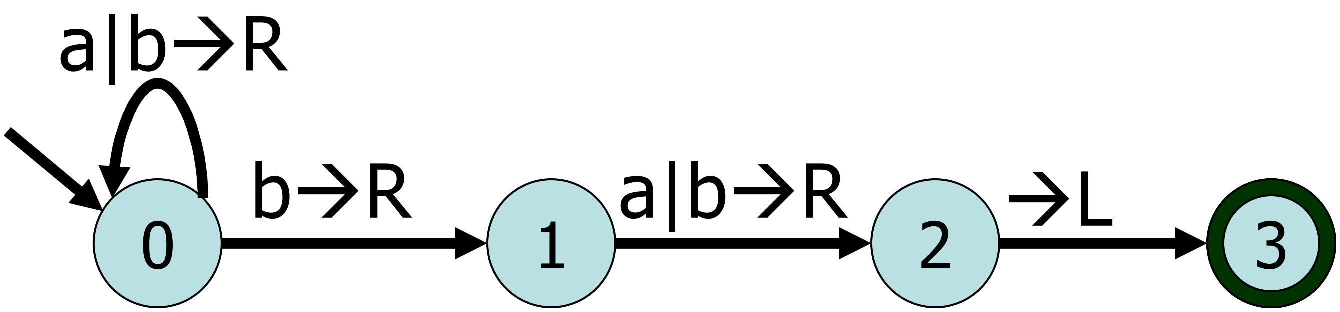
DEF: O ***tempo de execução não determinista*** de uma máquina de Turing não determinista N é a função $f(n)$ que dá o número máximo de transições que N efetua até parar (ou falhar) em qualquer ramo da computação, dada uma entrada arbitrária de comprimento n.

Q: Quando N é dito um decisor?

Tempo de Execução Não Determinista

R: Quando *f* (*n*) é finita para qualquer *n*.

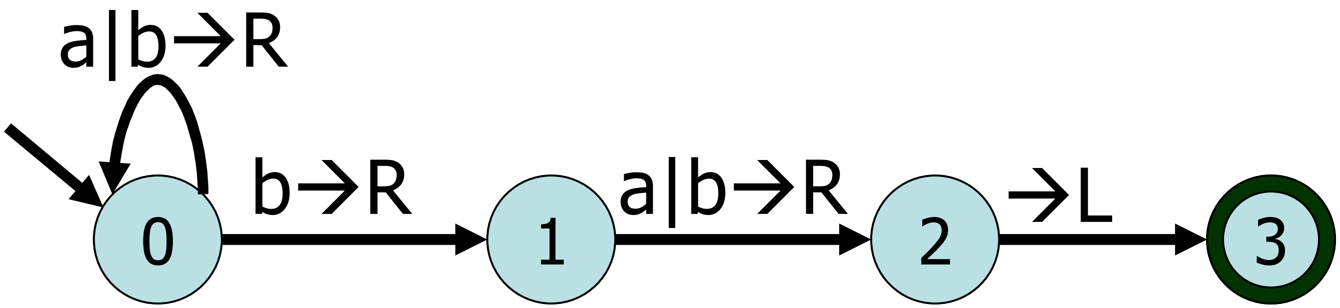
Considere a NTM decisora p/ $(a \cup b)^* b(a \cup b)$ (vista anteriormente):



Q: Qual é o tempo de execução?

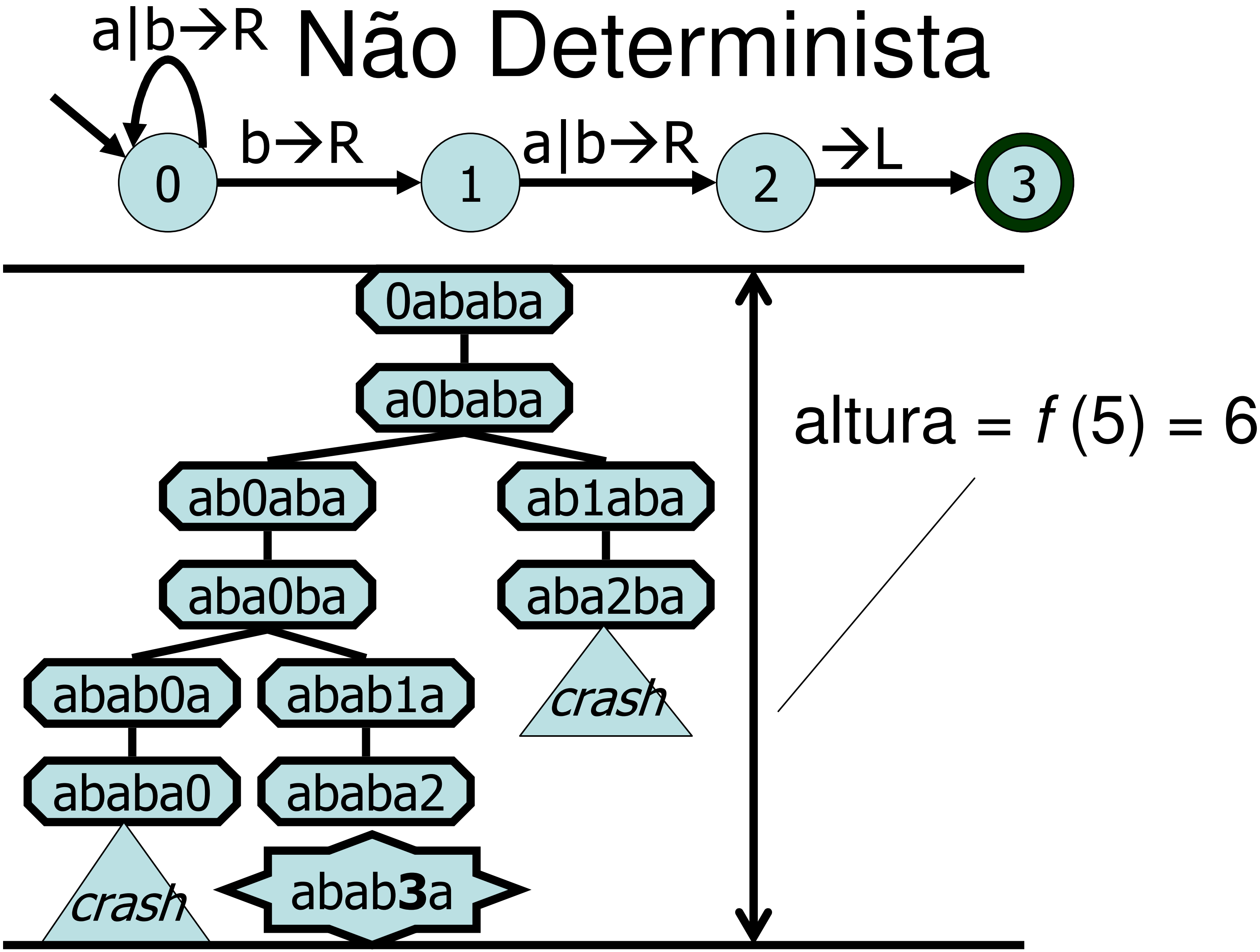
Tempo de Execução Não Determinista

R: $f(n) = n + 1$. Como todas as transições movem para a direita, o maior ramo possível é o ramo de aceitação, que lê toda a entrada e então pára.



Também podemos ver isso examinando a árvore de computação. O tempo de execução é a *altura da árvore*. EX:

Tempo de Execução Não Determinista



Classes de Complexidade de Tempo Não Determinista

Podemos definir classes de linguagens com base em TM's não deterministas, tal como fizemos para TM's deterministas.

DEF: Seja $g(n)$ uma função real. A ***classe de complexidade de tempo não determinista*** NTIME($g(n)$) consiste de todas as linguagens que são decididas por uma NTM em tempo de execução $O(g(n))$. Tais linguagens são ditas de ***complexidade tempo não determinista*** $g(n)$.

Q: Dê uma função $g(n)$ tal que o exemplo anterior esteja em NTIME($g(n)$).

A Classe **NP**

R: $g(n) = n$.

Esse é um exemplo de TM de *tempo polinomial não determinista,* tal como definido a seguir.

NP é a classe de linguagens que são decididas por uma NTM de complexidade de tempo polinomial. I.e.:

DEF 1: $\mathbf{NP} = \bigcup_{k=0}^{\infty} \mathrm{NTIME}(n^k)$

A Classe **NP**
Definições Alternativas

Alternativamente, **NP** é a classe de linguagens cujas instâncias positivas se pode *provar* que pertencem à linguagem por uma prova de tamanho polinomial. Embora uma prova possa ter tamanho arbitrariamente longo, apenas nos interessam provas curtas.

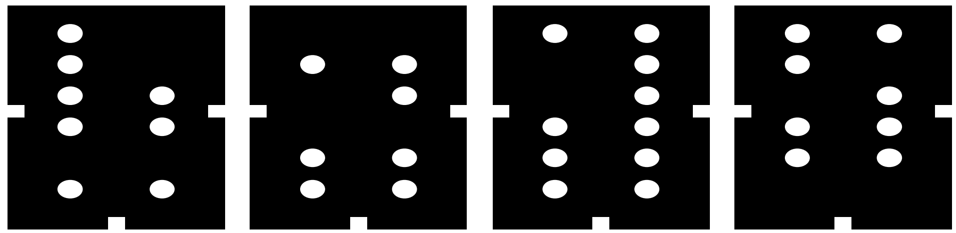
Q: De acordo com essa definição, o jogo de cartões perfurados seria **NP**?

Exemplo: Jogo de cartões Prova Curta

R: Sim. Se *for encontrada* uma solução para o jogo de cartões perfurados, podemos rapidamente provar que a solução dada é de fato uma solução.

EX: No caso do exemplo anterior, de 4 cartões, a prova seria do seguinte modo:

Exemplo: Jogo de cartões
Prova Curta



C1 C2 C3 C4

CLAIM: Uma solução é C1, C2, C3, flip(C4).

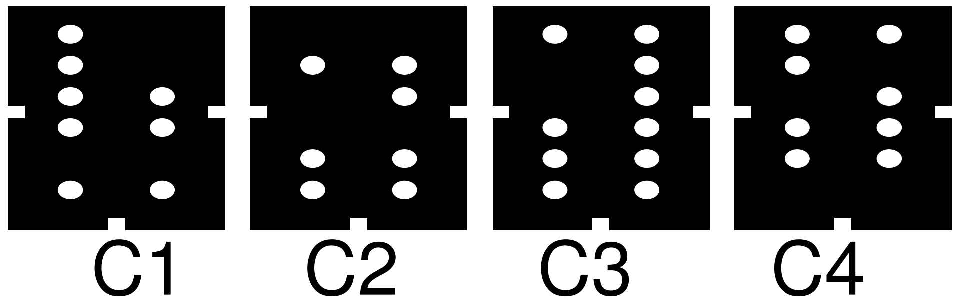
1) C2 cobre (1,L)
2) C2 cobre (1,R)
3) C3 cobre (2,L)

… *e assim por diante* …

12) flip(C4) cobre (6,R)

Q:Qual o tamanho da prova p/ *n* cartões c/*m linhas*?

Exemplo: Jogo de cartões Prova Curta



R: Para *m* linhas, são necessárias 2*m+1* linhas na prova. A primeira linha tem comprimento *O* (*n*). Todas as demais linhas são *O*(1). Portanto, o tamanho total da prova é *O*(*m*+*n*). O tamanho da entrada do problema é também *O*(*m*+*n*) pois podemos descrever cada cartão com *O*(*m*) caracteres. Portanto a prova tem *tamanaho linear*, e o jogo está em **NP**.

Q: Como provar que um jogo *não tem* solução?

A Classe Co-**NP**

R: Esse problema parece ser bem mais difícil. Parece não haver uma prova sistemática de que não existe solução, a não ser tentar todas as possíveis combinações flip/no-flip e verificar se restam buracos em cada caso. O problema de decidir se um jogo não tem solução está em Co-**NP**:

DEF: A classe Co-**NP** consiste de todas as linguagens cujos complementos estão em **NP**.

Jogo de cartões perfurados Algoritmo Não Determinista

R:

```
NondeterministicSolvePuzzle(cards C1, C2,…,Cn)
   for(i = 1 to n)
                flip Ci
        OR    skip      //nondeterministic "OR"
   put cards on pegs
   if (no holes) ACCEPT
   REJECT
```

Q: Qual é o tempo de execução?

Jogo de cartões perfurados Algoritmo Não Determinista

R: O tempo de execução é $O(m + n)$ não determinista em uma RAM, uma vez que temos um for-loop de tamanho n, em que cada passo é $O(1)$, e verificar se todos os buracos estão tapados é $O(m)$. Então, em termos da entrada, o algoritmo não determinista executa em tempo linear em uma RAM e, portanto, em tempo polinomial em uma NTM.

Redução de Tempo Polinomial

Lembre-se que uma linguagem *A* é Turing redutível para uma linguagem *B* se podemos construir um decisor para *A* supondo que existe um decisor para *B*. Quando lidamos com complexidade, queremos também garantir que a redução executa em tempo polinomial:

DEF: Se existe uma função computável de tempo polinomial *f* tal que $f(A) \subseteq B$ e $f(\overline{A}) \subseteq \overline{B}$ então *A* ***reduz por mapeamento polinomial*** para *B.*

Analogamente, podemos definir redução de co-mapeamento de tempo polinomial.

Redução de Tempo Polinomial
Notação

- *A* ***reduz por mapeamento de tempo polinomial*** para *B*:

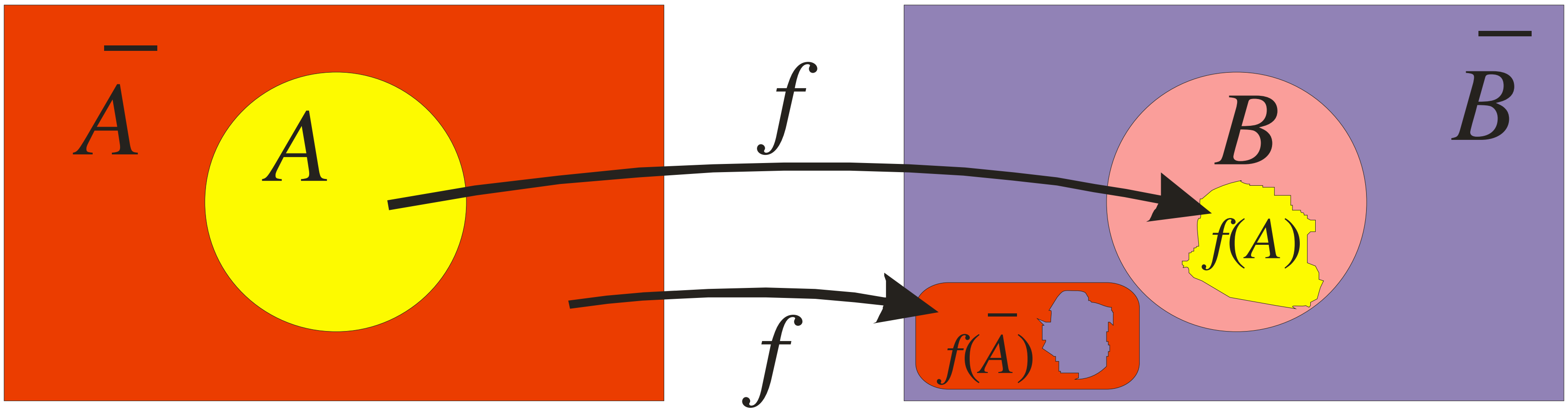
$$A \leq_P B$$

- *A* ***reduz por co-mapeamento de tempo polinomial*** para *B*:

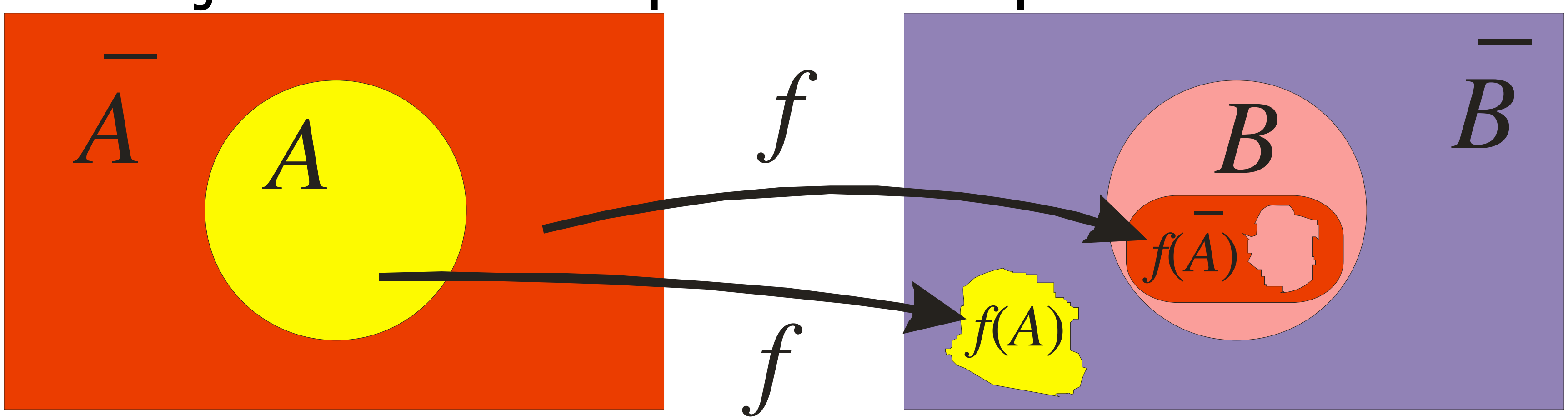
$$A \leq_P \overline{B}$$

Visualizando Reduções de Tempo Polinomial

Redução de mapeamento polinomial:



Redução de co-mapeamento polinomial:



Lemas sobre Redução de Tempo Polinomial

LEMA: Sejam A e B linguagens.

- Se $A \leq_P B$ e $B \in$ **P**, então $A \in$ **P**
- Se $A \leq_P \overline{B}$ e $B \in$ **P**, então $A \in$ **P**
- Se $A \leq_P B$ e $B \in$ **NP**, então $A \in$ **NP**
- Se $A \leq_P \overline{B}$ e $B \in$ **NP**, então $A \in$ co-**NP**
- Se $A \leq_P \overline{B}$ e $B \in$ co-**NP**, então $A \in$ **NP**

Provar esses fatos consiste em mostrar que reduções de tempo polinomial podem ser compostas com algoritmos de tempo polinomial (det[c] ou nãodet[c]) resultando algoritmos de tempo polinomial.

Lemas sobre Redução de Tempo Polinomial

Lema de Transitividade. Sejam *A* e *B* linguagens.

- Se $A \leq_P B$ e $B \leq_P C$ então $A \leq_P C$
- Se $A \leq_P \overline{B}$ e $B \leq_P \overline{C}$ então $A \leq_P C$

A composição de reduções de tempo polinomial é uma redução de tempo polinomial porque polinômio(polinômio) é um polinômio.

SAT

O mais famoso problema em **NP** é o problema de satisfazibilidade SAT:

Dada: uma fórmula ϕ da lógica booleana, envolvendo variáveis e os connectivos $\wedge$, $\neg$, $\vee$.

Decidir: ϕ é satisfazível? I.e., existe uma atribuição de valores "true" ou "false" para as variáveis que torna ϕ verdadeira?

EX: $(x \wedge \neg x) \vee \neg y$ é uma instância de SAT mas $x \wedge \neg x$ não é.

Interessante: *todo* problema da classe **NP** pode ser reduzido para SAT em tempo polinomial!

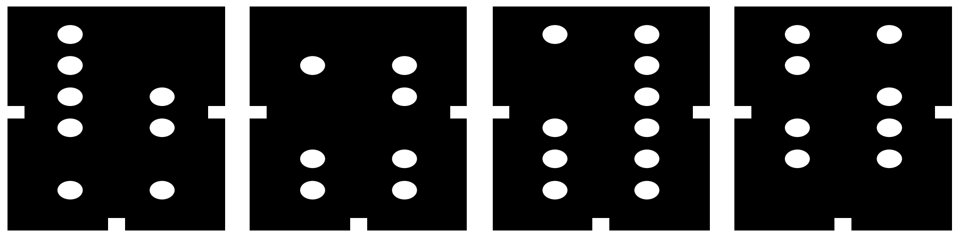
Vejamos como é isso para o jogo de cartões.

Reduzindo o Jogo de Cartões para SAT

Para reduzir o problema para SAT precismos pensar o que são as variáveis e como elas devem ser usadas para converter cartões perfurados em fórmulas.

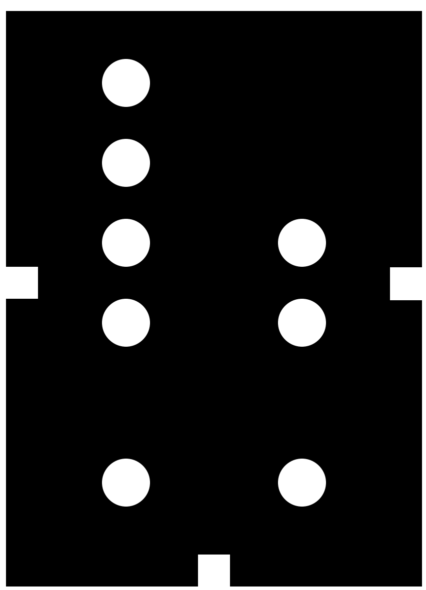
IDÉIA: Crie uma subfórmula para cada possivel furo. Cada cartão corresponde a uma variável que, se "true", satisfaz as fórmulas correspondentes aos seus furos faltantes. Virar o cartão torna a variável "false", satisfazendo então as outras fórmulas:

Reduzindo o Jogo de Cartões para SAT



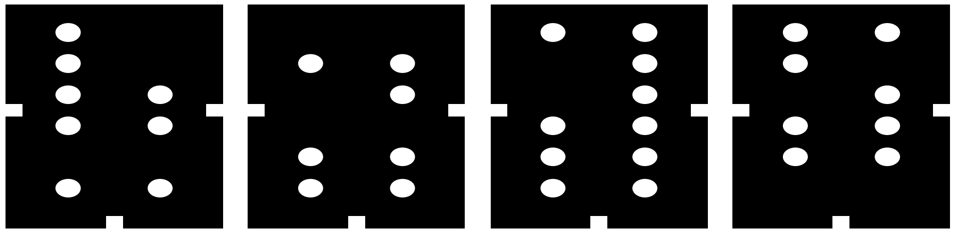
C1 C2 C3 C4

Primeiro cartão cria a tabela de fórmulas:



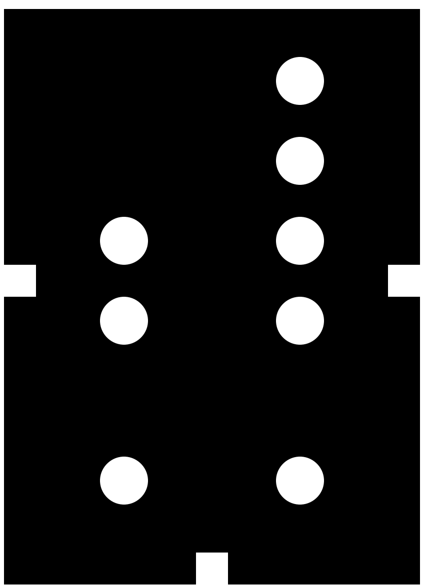
	x_1
	x_1
x_1	x_1

Reduzindo o Jogo de Cartões para SAT



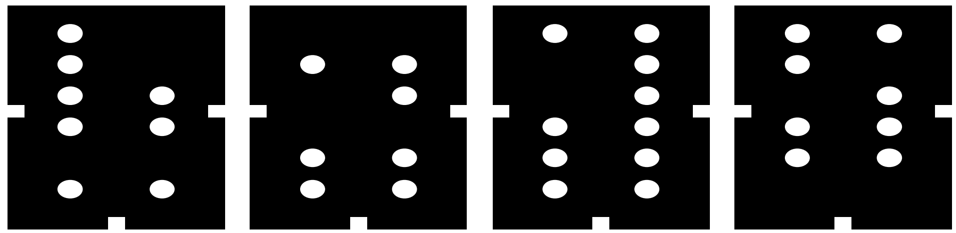
C1 C2 C3 C4

Primeiro cartão *virado* adiciona à tabela:



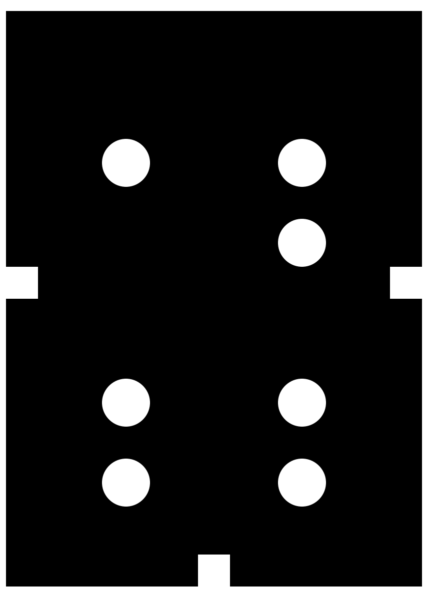
$\neg x_1$	x_1
$\neg x_1$	x_1
$x_1 \vee \neg x_1$	$x_1 \vee \neg x_1$

Reduzindo o Jogo de Cartões para SAT



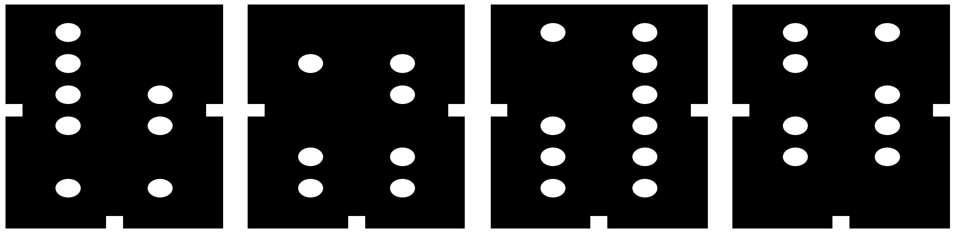
C1 C2 C3 C4

Segundo cartão adiciona:



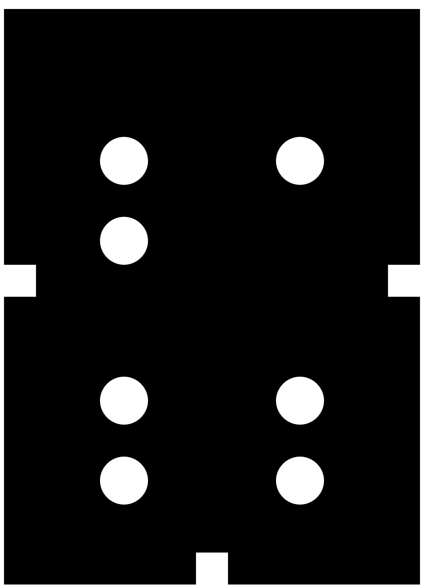
$\neg x_1 \vee x_2$	$x_1 \vee x_2$
$\neg x_1$	x_1
x_2	
x_2	x_2
$x_1 \vee \neg x_1$	$x_1 \vee \neg x_1$

Reduzindo o Jogo de Cartões para SAT



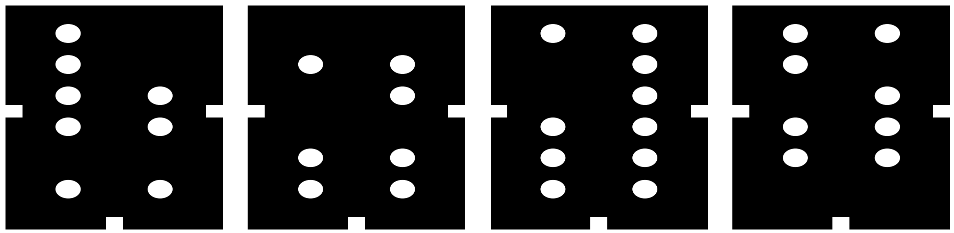
C1 C2 C3 C4

Segundo cartão *virado* adiciona :



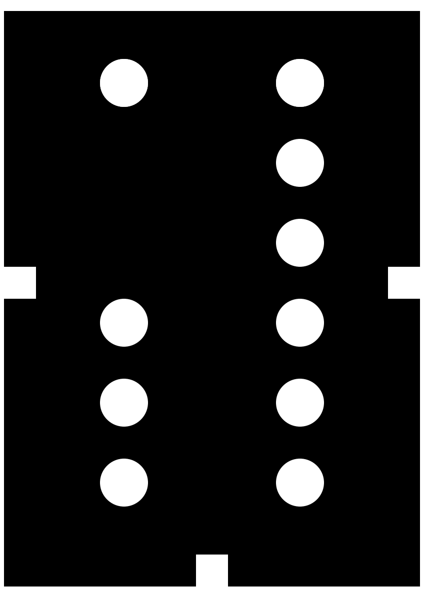
$\neg x_1 \vee x_2 \vee \neg x_2$	$x_1 \vee x_2 \vee \neg x_2$
$\neg x_1$	x_1
x_2	$\neg x_2$
$x_2 \vee \neg x_2$	$x_2 \vee \neg x_2$
$x_1 \vee \neg x_1$	$x_1 \vee \neg x_1$

Reduzindo o Jogo de Cartões para SAT



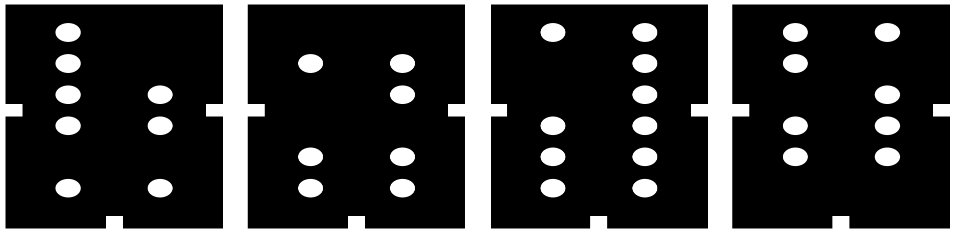
C1 C2 C3 C4

Terceiro cartão adiciona:



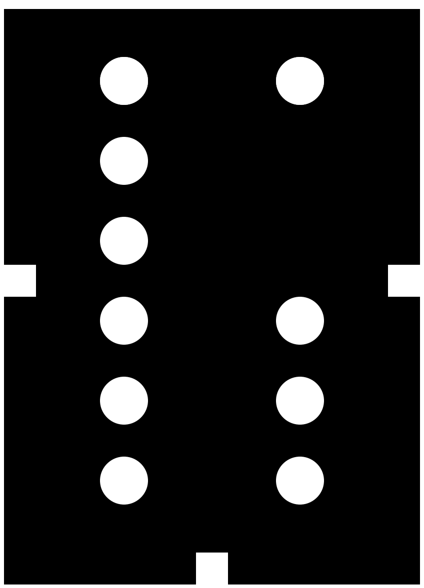
$\neg x_1 \vee x_2 \vee \neg x_2$	$x_1 \vee x_2 \vee \neg x_2$
$\neg x_1 \vee x_3$	x_1
$x_2 \vee x_3$	$\neg x_2$
$x_2 \vee \neg x_2$	$x_2 \vee \neg x_2$
$x_1 \vee \neg x_1$	$x_1 \vee \neg x_1$

Reduzindo o Jogo de Cartões para SAT



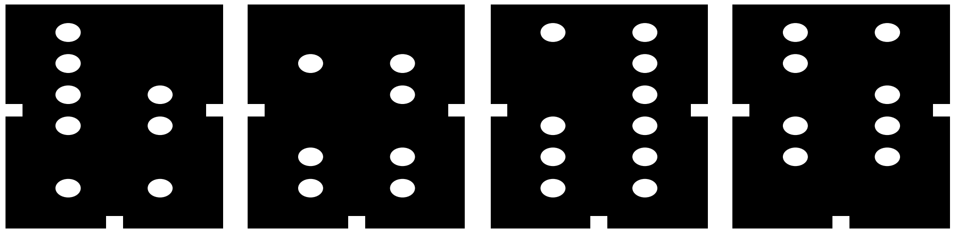
C1 C2 C3 C4

Terceiro cartão virado adiciona:



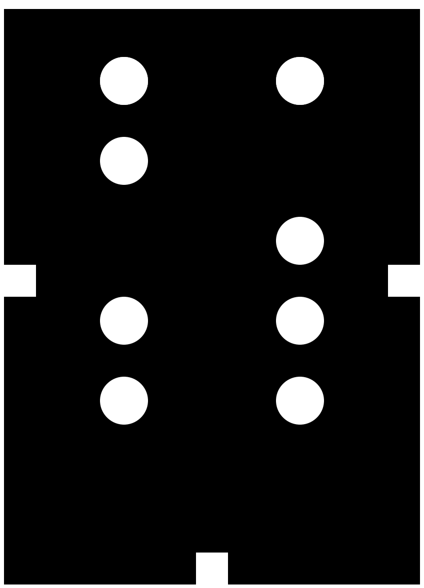
$\neg x_1 \lor x_2 \lor \neg x_2$	$x_1 \lor x_2 \lor \neg x_2$
$\neg x_1 \lor x_3$	$x_1 \lor \neg x_3$
$x_2 \lor x_3$	$\neg x_2 \lor \neg x_3$
$x_2 \lor \neg x_2$	$x_2 \lor \neg x_2$
$x_1 \lor \neg x_1$	$x_1 \lor \neg x_1$

Reduzindo o Jogo de Cartões para SAT



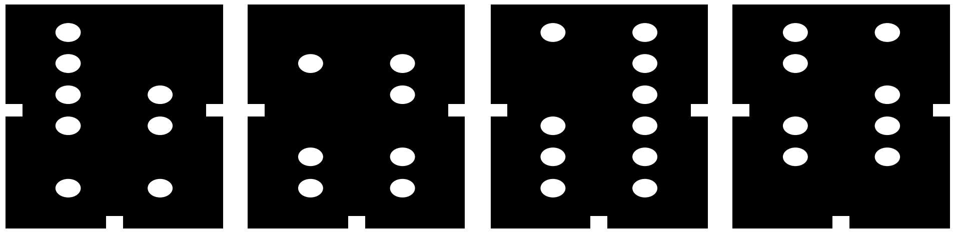
C1 C2 C3 C4

Quarto cartão adiciona:



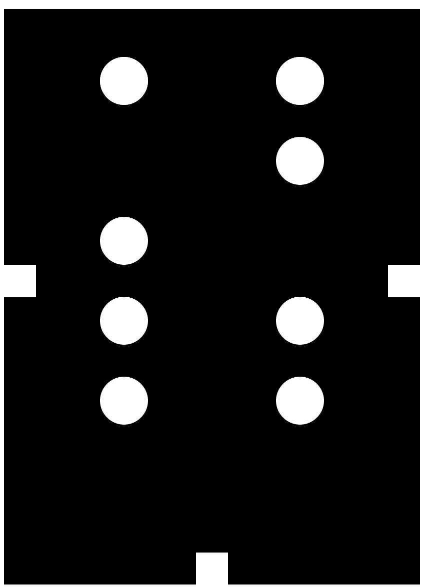
$\neg x_1 \vee x_2 \vee \neg x_2$	$x_1 \vee x_2 \vee \neg x_2$
$\neg x_1 \vee x_3$	$x_1 \vee \neg x_3 \vee x_4$
$x_2 \vee x_3 \vee x_4$	$\neg x_2 \vee \neg x_3$
$x_2 \vee \neg x_2$	$x_2 \vee \neg x_2$
$x_1 \vee \neg x_1$	$x_1 \vee \neg x_1$
x_4	x_4

Reduzindo o Jogo de Cartões para SAT



C1 C2 C3 C4

Quarto cartão virado adiciona:



$\neg x_1 \vee x_2 \vee \neg x_2$	$x_1 \vee x_2 \vee \neg x_2$
$\neg x_1 \vee x_3 \vee \neg x_4$	$x_1 \vee \neg x_3 \vee x_4$
$x_2 \vee x_3 \vee x_4$	$\neg x_2 \vee \neg x_3 \vee \neg x_4$
$x_2 \vee \neg x_2$	$x_2 \vee \neg x_2$
$x_1 \vee \neg x_1$	$x_1 \vee \neg x_1$
$x_4 \vee \neg x_4$	$x_4 \vee \neg x_4$

Reduzindo o Jogo de Cartões para SAT

Perguntar se é possível tapar todos os furos é o mesmo que perguntar se é possível satisfazer todas as fórmulas de uma vez.

$\neg x_1 \vee x_2 \vee \neg x_2$	$x_1 \vee x_2 \vee \neg x_2$
$\neg x_1 \vee x_3 \vee \neg x_4$	$x_1 \vee \neg x_3 \vee x_4$
$x_2 \vee x_3 \vee x_4$	$\neg x_2 \vee \neg x_3 \vee \neg x_4$
$x_2 \vee \neg x_2$	$x_2 \vee \neg x_2$
$x_1 \vee \neg x_1$	$x_1 \vee \neg x_1$
$x_4 \vee \neg x_4$	$x_4 \vee \neg x_4$

I.e., se a seguinte conjunção é satisfazível:

Reduzindo o Jogo de Cartões para SAT

$(\neg x_1 \vee x_2 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_2) \wedge (x_2 \vee \neg x_2) \wedge (x_1 \vee \neg x_1) \wedge (x_1 \vee \neg x_1) \wedge (x_4 \vee \neg x_4) \wedge (x_4 \vee \neg x_4)$

De fato, atribuindo "true" às três primeiras variáveis e "false" à quarta a fórmula é satisfeita, já que o jogo pode ser resolvido mantendo os três primeiros cartões em posição normal e com o quarto cartão virado.

Variantes de SAT -CSAT

Note que a fórmula obtida é basicamente uma *conjunção* (AND's) de *disjunções* (OR's). Para resolver o jogo de cartões seria suficiente saber resolver SAT apenas para tais expressões.

DEF: Um ***literal*** é uma variável tal como x ou sua negação $\neg x$. Uma ***cláusula*** é uma disjunção de literais (e.x. $x_1 \vee \neg x_3 \vee x_4$). Uma fórmula booleana é uma ***fórmula normal conjuntiva*** (or ***cnf***) se é uma conjunção de cláusulas.

CSAT é o seguinte problema:

Dada: Uma cnf ϕ.

Decidir: ϕ é satisfazível?

Variantes de SAT
-*n*SAT

DEF: Uma fórmula booleana é *n-**cnf*** se é uma conjunção de cláusulas, cada qual com no máximo n literais.

*n*SAT é o seguinte problema:

Dada: Uma *n*-cnf ϕ.

Decidir: ϕ é satisfazível?