Computabilidade

Aula 4 – Máquina de Turing e o Problema da Parada

Prof. Rodrigo Mafort

Últimas aulas

- Revisão de Complexidade Computacional
- Revisão de Autômatos
- Máquinas de Turing

Máquina de Turing

- Proposta por Alan Turing em 1936

- A máquina de Turing é um autômato que usa uma fita para armazenar as informações

- Essa fita pode ser lida, escrita e movimentada para a esquerda e para a direita.

- A fita possibilita uma memória infinita para a máquina e assegura a habilidade de ler as entradas mais de uma vez. Permite também sobrescrever valores das entradas.

Autômatos Finitos vs Máquina Turing

- Os autômatos de pilha possuem uma fita com a entrada. Essa fita pode ser lida e a cada leitura a fita se movimenta para a próxima entrada. A pilha é usada como uma memória auxiliar pelo autômato.

- Nas máquinas de Turing a entrada e a memória formam uma única fita. Os dados da entrada são escritos na fita, que pode ser alterada e movimentada livremente (para esquerda e para direita) pela máquina.

- Ao permitir escrever na fita, a Máquina de Turing permite alterar os dados codificados na fita.

Autômato Finitos vs Máquina Turing

- Uma máquina de Turing pode ler e escrever na fita
- A cabeça de leitura pode ser mover para a esquerda e para a direita
- A fita é infinita

Máquina de Turing

- Composta por três partes:
    - Fita
    - Unidade de Controle
    - Programa

Definição formal

- Uma máquina de Turing é definida por um 7-upla:
- $M = \left(Q, \Sigma, \Gamma, Q, \delta, q_0, q_{aceita}, q_{rejeita}\right)$
- Onde:
  - Q é um conjunto finito de estados possíveis para a máquina
  - Σ é o alfabeto de símbolos de entrada (sem o símbolo de branco β)
  - Γ é o alfabeto da fita ($\beta \in \Gamma$ e $\Sigma \subseteq \Gamma$)
  - δ é o programa da máquina: $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{E, D\}$
  - q_0 é o estado inicial da máquina (ao ser iniciada, ela começa no estado q_0)
  - q_{aceita} é o estado de aceitação da máquina ($q_{aceita} \in Q$)
  - $q_{rejeita}$ é o estado de rejeição da máquina ($q_{rejeita} \in Q$ e $q_{rejeita} \neq q_{aceita}$)

- Diferente dos autômatos finitos, os estados de aceitação e rejeição fazem efeito imediatamente: a máquina para ao entrar em um desses estados.

Programa do Autômato

- δ é o programa da máquina (função de transição):
- $q_i \times \gamma_a \rightarrow \; q_j \times \gamma_b \times \; s$

- Se
  - a máquina está no estado $q_i \in Q$ e
  - lê o símbolo $\gamma_a \; \in \; \Gamma$ da fita

- Então:
  - vá para o estado $q_j \in Q$ e
  - escreva o símbolo $\gamma_b \; \in \; \Gamma$ na fita e
  - mova a fita no sentido $s \; \in \{E, D\}$ (esquerda ou direita)

Atividade Proposta

- Suponha uma fita que é formada por dois números na base binária separadas por um sinal de #.

- Como escrever um “programa” capaz de verificar se os dois números são iguais usando uma máquina de Turing?

- Pergunta 1: Qual é o alfabeto de entrada Σ na fita?
- Pergunta 2: Quais símbolos precisamos ler ou escrever na fita (Γ) além de Σ e β (branco)?
- Pergunta 3: Quais e quantos estados (além de q_0, q_{aceita} e $q_{rejeita}$) vamos precisar?
- Pergunta 4: Como definir as transições (o programa)?

Ideias:

- Pergunta 1: Qual é o alfabeto de entrada Σ na fita?
  - $\Sigma = \{0,1,\#\}$

- Pergunta 2: Quais símbolos precisamos ler ou escrever na fita além de Σ e β (branco)?
  - $\Gamma = \{\beta, \$, 0,1, \#\}$ (trocar símbolos já lidos e processados por $)

- Pergunta 3: Quais e quantos estados (além de q_0, q_{aceita} e $q_{rejeita}$) vamos precisar?
  - Meu modelo tem 9 estados (inicial, aceitação, rejeição e outros 6)

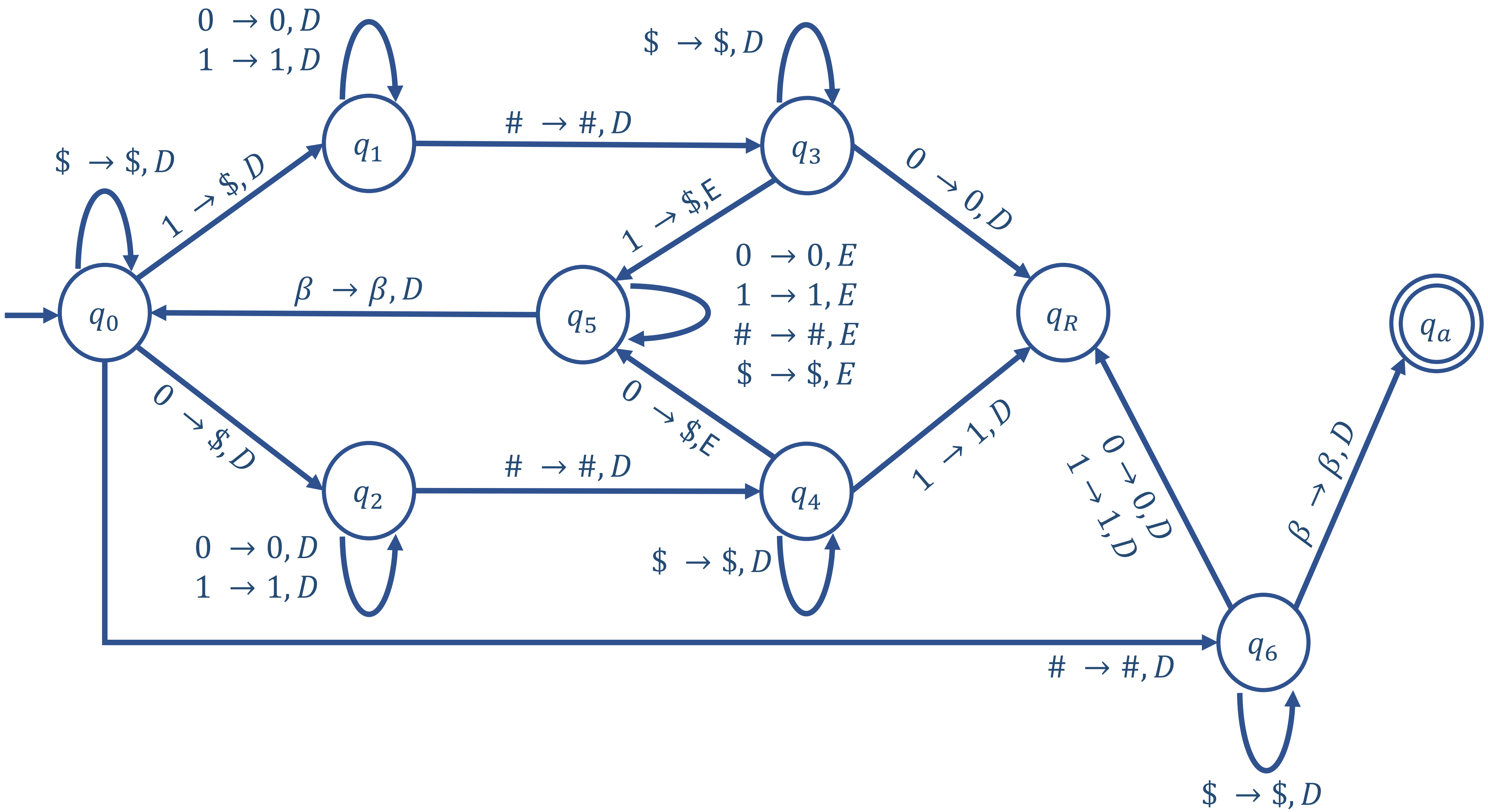
- Pergunta 4: Como definir as transições (o programa)?
  - O que fazer ao encontrar um 0 ou 1 no número da esquerda?
  - Como buscar o mesmo dígito no número da direita?

Teste seu modelo

- https://turingmachinesimulator.com/

- init: q0 — o estado inicial
- accept: qaceita — o estado de aceitação

- q0,$ — estando no estado q0, se ler $
- q0,$,> — vá para o estado q0, escreva $ e se desloque para direita



q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_R
q_a
$\$ \rightarrow \$, D$
$0 \rightarrow 0, D$
$1 \rightarrow 1, D$
$1 \rightarrow \$, D$
$0 \rightarrow \$, D$
$\# \rightarrow \#, D$
$\# \rightarrow \#, D$
$\$ \rightarrow \$, D$
$0 \rightarrow 0, D$
$1 \rightarrow 1, D$
$\$ \rightarrow \$, D$
$1 \rightarrow \$, E$
$0 \rightarrow 0, D$
$0 \rightarrow 0, E$
$1 \rightarrow 1, E$
$\# \rightarrow \#, E$
$\$ \rightarrow \$, E$
$\beta \rightarrow \beta, D$
$0 \rightarrow \$, E$
$1 \rightarrow 1, D$
$0 \rightarrow 0, D$
$1 \rightarrow 1, D$
$\beta \rightarrow \beta, D$
$\# \rightarrow \#, D$
$\$ \rightarrow \$, D$

Máquinas de Turing e Computadores

- O conceito de máquina de Turing é a base na qual os computadores foram construídos.

- Todo computador é uma máquina de Turing?
  - +/- ....
  - Máquina de Turing tem memória infinita (fita infinita)
  - Computadores tem memória finita

- Tudo o que um computador pode fazer uma máquina de Turing também pode.

Algoritmos e Máquinas de Turing

- Em 1900, um matemático chamado David Hilbert disse que todo problema matemático bem formulado por ser resolvido por uma sequência bem estruturada de passos.

- Ele deu um exemplo: É possível escrever uma sequência de instruções para verificar se um polinômio apresenta ou não uma raiz inteira.

- Hoje sabemos que essa afirmação é falsa. Mas para demonstrar isso foi necessário definir formalmente o que é um algoritmo. Em 1936, Alonzo Church e Alan Turing definiram formalmente (ao mesmo tempo usando abordagens diferentes) o que são algoritmos.

Tese de Church-Turing

- **A noção intuitiva de um algoritmo é igual aos algoritmos de Máquinas de Turing.**
- Isso é:
  - Todo algoritmo (como conhecemos hoje) apresenta uma versão equivalente em uma Máquina de Turing.
  - Em outras palavras, todo algoritmo pode ser transformado em um conjunto de estados e funções de transições (um algoritmo) da Máquina de Turing.

- Esse conceito permitiu demonstrar os limites da computação.

- Isso é: problemas que não podem ser tratados computacionalmente (não é possível definir um algoritmo para resolver esse problema).
- Dois casos interessantes:
  - Problema da Parada (tema da próxima aula)
  - Em 1970 foi demonstrado que o problema das raízes inteiras dos polinômios não pode ser tratado computacionalmente (não é possível construir um algoritmo que resolva esse problema).

Antes de Continuar

- Projete uma máquina de Turing para verificar se uma entrada satisfaz a seguinte regra: $\{a^n b^n c^n\}$.

- Em outras palavras: projeto um autômato que reconheça se uma entrada pertence a linguagem $\{a^n b^n c^n\}$.

Definição formal

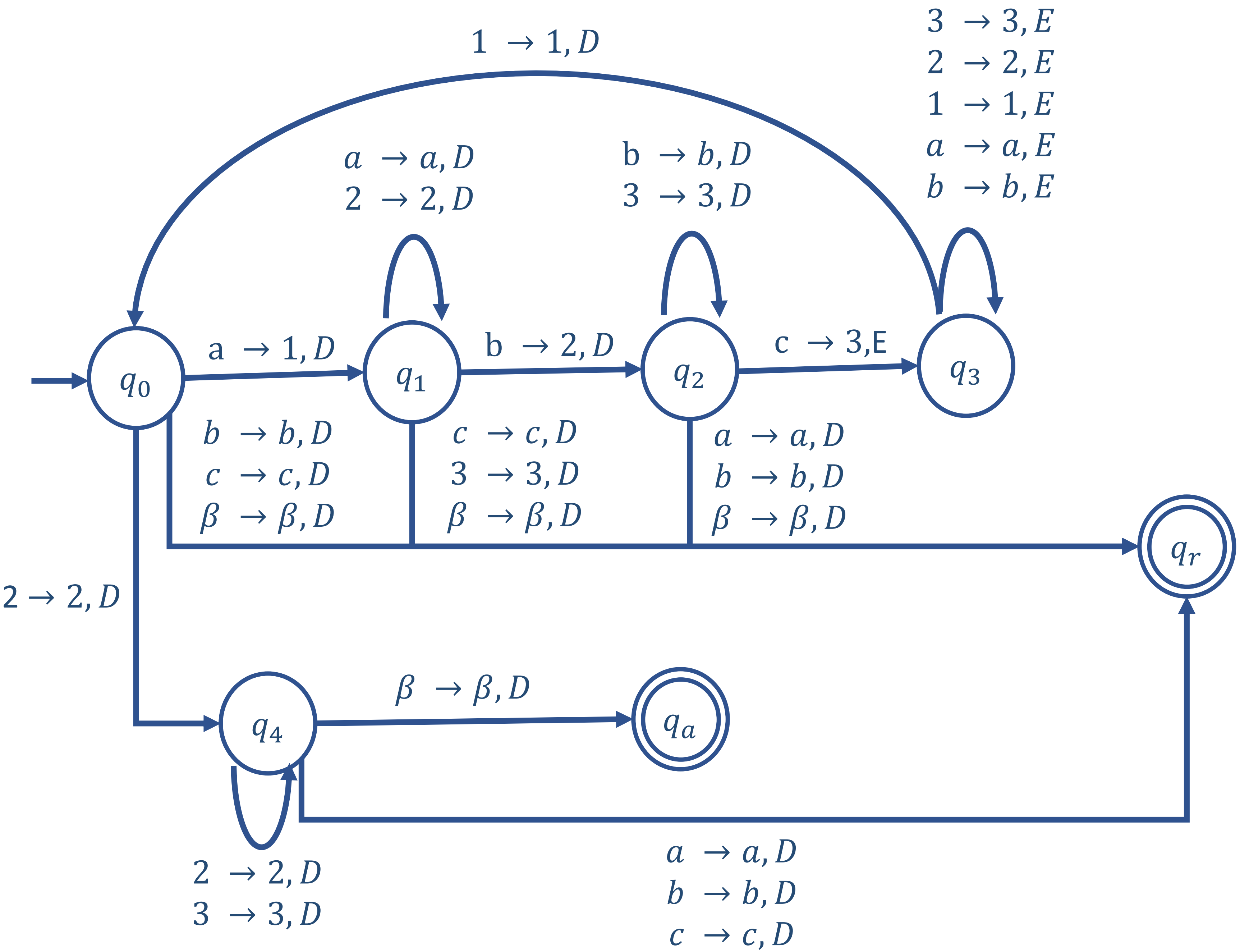
- $M = (Q, \Sigma, \Gamma, Q, \delta, q_0, q_{aceita}, q_{rejeita})$
- Onde:
  - Q é um conjunto finito de estados possíveis para a máquina
  - Σ é o alfabeto de símbolos de entrada (sem o símbolo de branco β)
  - Γ é o alfabeto da fita ($\beta \in \Gamma$ e $\Sigma \subseteq \Gamma$)
  - δ é o programa da máquina: $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{E, D\}$
  - q_0 é o estado inicial da máquina (ao ser iniciada, ela começa no estado q_0)
  - q_{aceita} é o estado de aceitação da máquina ($q_{aceita} \in Q$)
  - $q_{rejeita}$ é o estado de rejeição da máquina ($q_{rejeita} \in Q$ e $q_{rejeita} \neq q_{aceita}$)

- Diferente dos autômatos finitos, os estados de aceitação e rejeição fazem efeito imediatamente: a máquina para ao entrar em um desses estados.

Definição formal

- $M = (Q, \Sigma, \Gamma, Q, \delta, q_0, q_{aceita}, q_{rejeita})$
- Onde:
  - $Q = \{q_0, \ \cdots \ , \ q_i, q_{aceita}, \ q_{rejeita}\}$
  - $\Sigma = \{a, b, c\}$
  - $\Gamma = \{a, b, c, \beta, 1, 2, 3\}$
  - δ é o programa da máquina

- Lembrar: diferente dos autômatos finitos, os estados de aceitação e rejeição fazem efeito imediatamente: a máquina para ao entrar em um desses estados.


1 → 1, D
3 → 3, E
2 → 2, E
1 → 1, E
a → a, E
b → b, E
a → a, D
2 → 2, D
b → b, D
3 → 3, D
q0
a → 1, D
q1
b → 2, D
q2
c → 3,E
q3
b → b, D
c → c, D
β → β, D
c → c, D
3 → 3, D
β → β, D
a → a, D
b → b, D
β → β, D
qr
2 → 2, D
q4
β → β, D
qa
2 → 2, D
3 → 3, D
a → a, D
b → b, D
c → c, D

Aceitação ou Rejeição

- A máquina de Turing **aceita** uma entrada se:
  1. Partindo de um estado inicial
  2. Em uma sequência finita de passos
  3. Chegamos ao estado de aceitação

- A máquina de Turing **rejeita** uma entrada se:
  1. Partindo de um estado inicial
  2. Em uma sequência finita de passos
  3. Chegamos ao estado de rejeição

- Se a máquina nunca entrar em um estado de aceitação ou de rejeição então a máquina entrou em loop ou simplesmente **não para**.

Tese de Church-Turing

- **A noção intuitiva de um algoritmo é igual aos algoritmos de Máquinas de Turing.**
- Isso é:
  - Todo algoritmo (como conhecemos hoje) apresenta uma versão equivalente em uma Máquina de Turing.
  - Em outras palavras, todo algoritmo pode ser transformado em um conjunto de estados e funções de transições (um algoritmo) da Máquina de Turing.
- Tudo que pode ser efetivamente calculado é computável.
- Computável => pode ser produzido/reconhecido por uma Máquina de Turing.
- Esse conceito permitiu demonstrar os limites da computação.

O Problema de Hilbert

- Em 1900, um matemático chamado David Hilbert disse que todo problema matemático bem formulado por ser resolvido por uma sequência bem estruturada de passos.

- Ele deu um exemplo:

- É possível escrever uma sequência de instruções para verificar se um polinômio apresenta ou não uma raiz inteira.

O Problema de Hilbert

- Em 1936, Alonzo Church e Alan Turing definiram formalmente (ao mesmo tempo usando abordagens diferentes) o que são algoritmos.

- $A = \{ < p > | \ p \text{ é } um\ polinômio\ com\ uma\ raiz\ inteira \}$

- Existe uma Máquina de Turing capaz de reconhecer A?
- A é Turing-Decidível?

- Em 1970: Foi demonstrado que não.

O Problema de Hilbert

- $\mathrm{B} = \{ < p > \mid \ p \text{ é } \mathit{um\ polinômio\ sobre\ x\ com\ uma\ raiz\ inteira} \}$

- Como verificar/reconhecer B? Isso é: como verificar se um polinômio sobre x apresenta ou não uma raiz inteira?

- Como resolver essa questão?

O Problema de Hilbert

- Como verificar se um polinômio sobre x apresenta ou não uma raiz inteira?

1. Testar cada possibilidade da seguinte sequência: $0, 1, -1, 2, -2, 3, -3, 4, -4, \ldots$
2. Se o polinômio resultou em 0: Ele apresenta uma raiz inteira (aceitar)
3. Se nenhuma possibilidade resultou em 0, então rejeite.

- Qual é o problema dessa ideia? O conjunto de possibilidades é infinito...

O Problema de Hilbert

- Como evitar o problema das infinitas possibilidades?

- Foi demonstrado que não é necessário testar todas as possibilidades.
- As possíveis raízes estão na no intervalo: $-k\frac{C_{max}}{C_1}, \cdots, 0, \cdots, k\frac{C_{max}}{C_1}$
- Onde:
  - k é o número de termos do polinômio
  - C_{max} é o coeficiente de maior valor absoluto
  - C_1 é o coeficiente do termo de maior ordem

- O Problema: Não é possível definir esses mesmo limites para polinômios com mais de uma variável.

O Problema de Hilbert

- Como não é possível testar o infinito de possibilidades e não é possível calcular um intervalo finito:

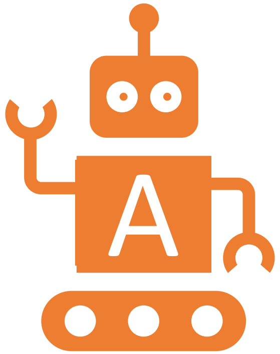
- Não é possível definir uma Máquina de Turing que verifique se um polinômio com mais de uma variável apresenta ou não raízes inteiras.

- E como existe uma associação entre Máquinas de Turing, algoritmos e computadores:
  - Não é possível resolver esse problema computacionalmente.

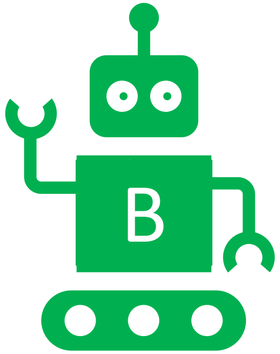
- Esse é um problema indecidível: é impossível construir um algoritmo que sempre responde corretamente sim ou não.

Problema da Parada

- Vamos supor duas máquinas A e B:



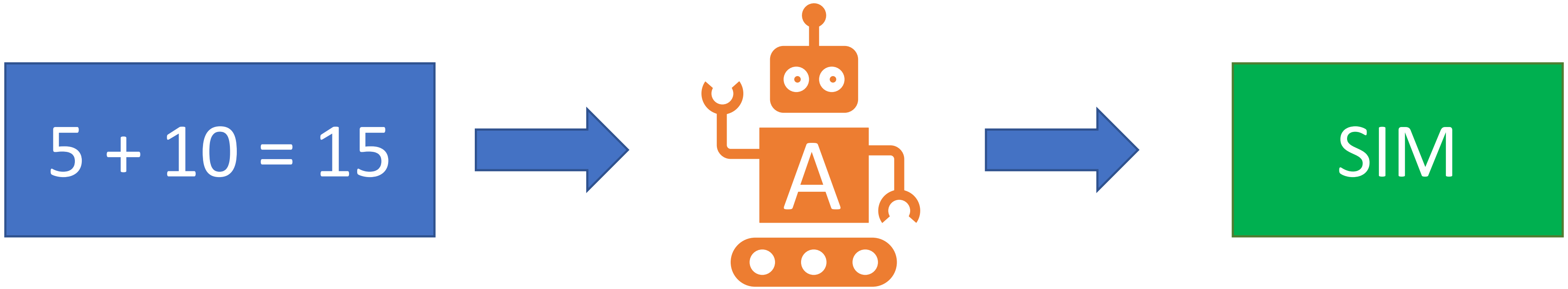
- A primeira recebe (Máquina A) como entrada expressões aritméticas
- Com base na entrada:
  - Se for uma expressão: A máquina funciona e retorna uma resposta (Para em um estado de aceitação ou rejeição)
  - Se for qualquer outra entrada: A máquina não funciona (Entra em Loop – Não para)



- A segunda (Máquina B) recebe como entrada polígonos
- Com base na entrada:
  - Se forem dois polígonos: A máquina funciona e retorna uma resposta
  - Se for qualquer outra entrada: A máquina não funciona (Entra em Loop – Não para)

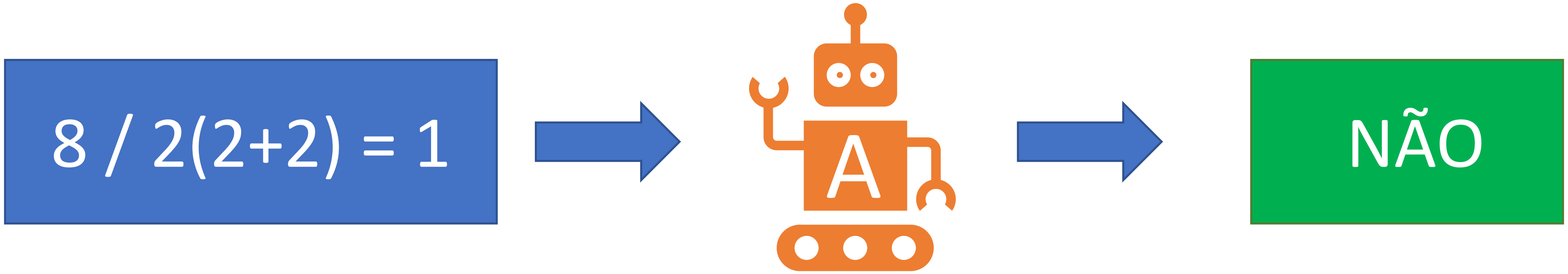
Problema da Parada

- Suponha uma máquina A que resolve problemas aritméticos:



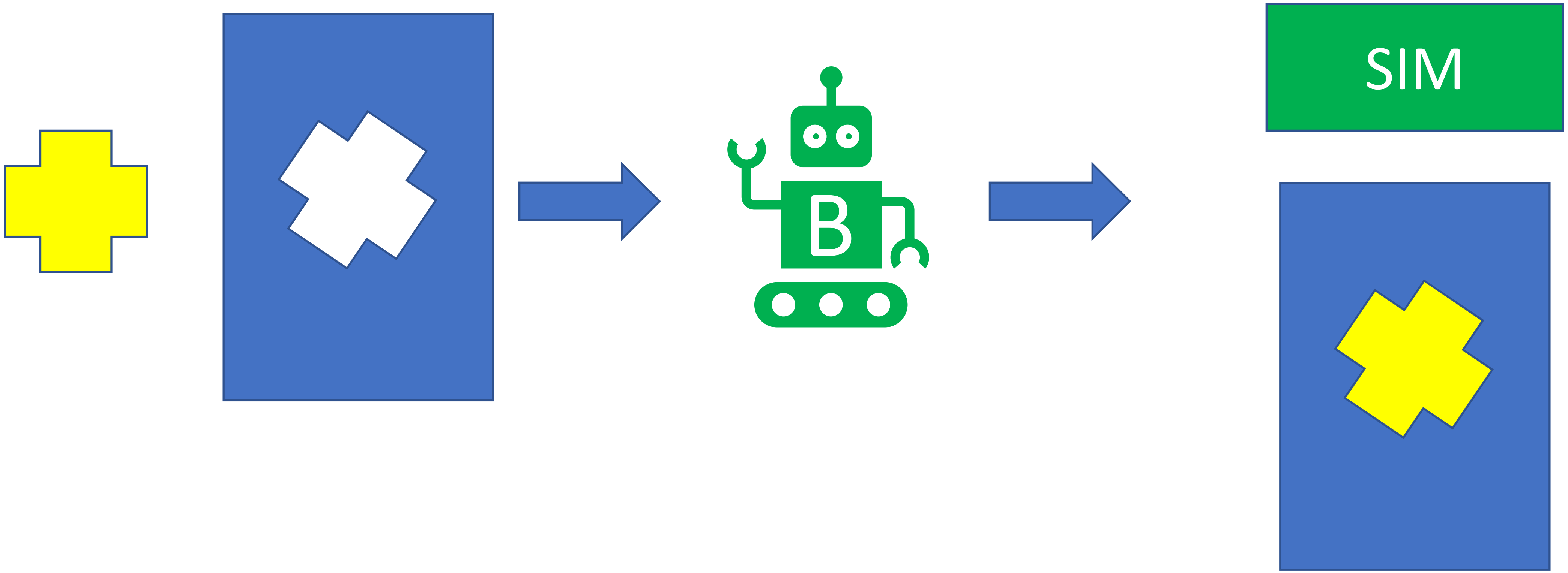
Problema da Parada

- Suponha uma máquina A que resolve problemas aritméticos:



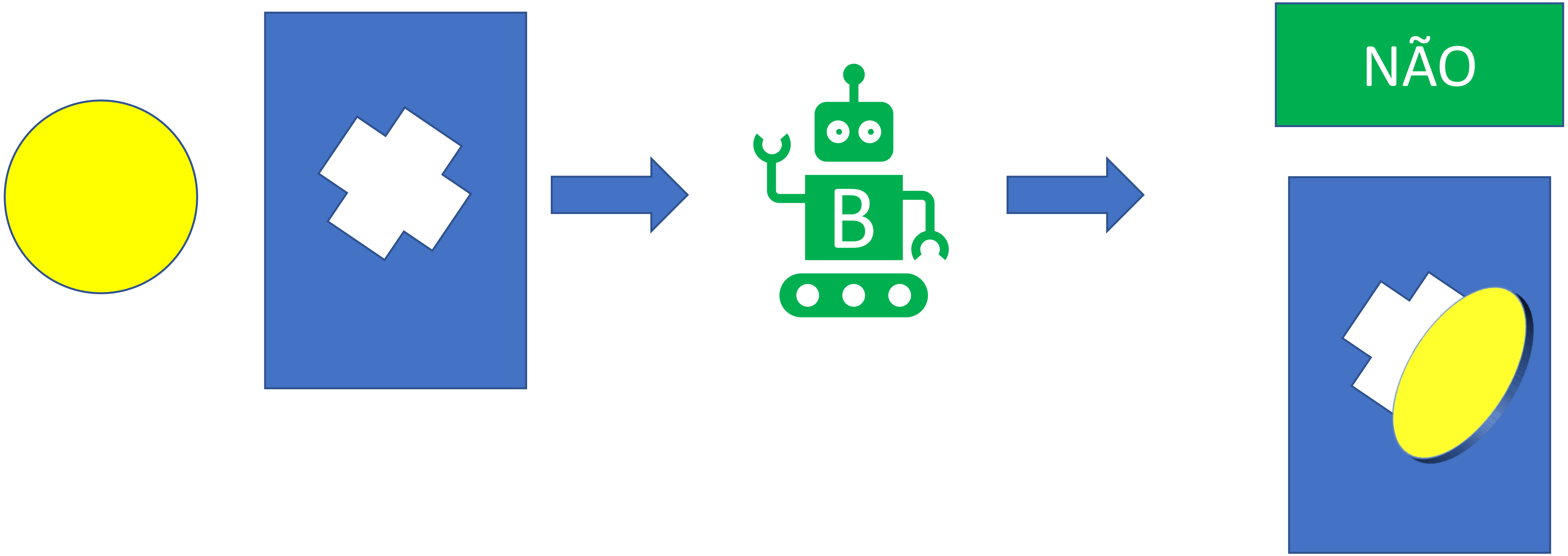
Problema da Parada

- Suponha agora uma máquina B que verifica se dois polígonos encaixam:



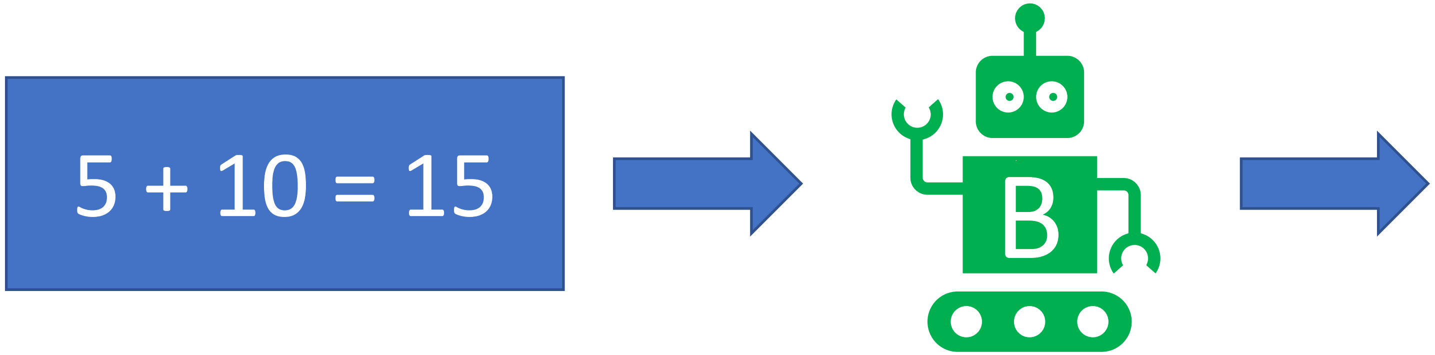
Problema da Parada

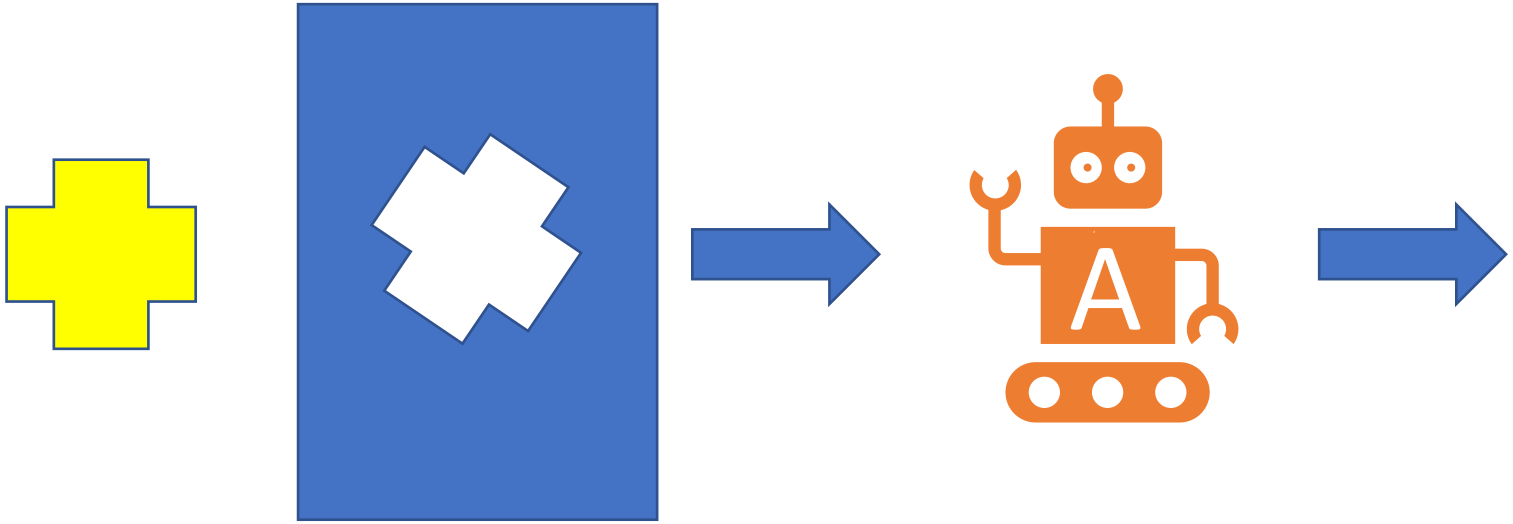
- Suponha agora uma máquina B que verifica se dois polígonos encaixam:



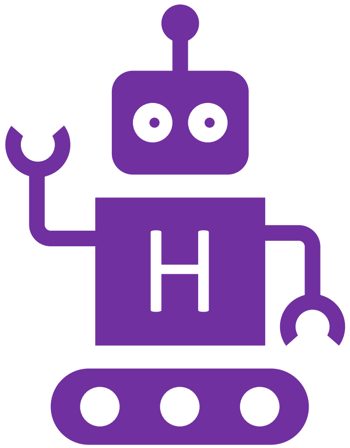
Problema da Parada

- O que acontece se trocarmos as entradas?





Problema da Parada



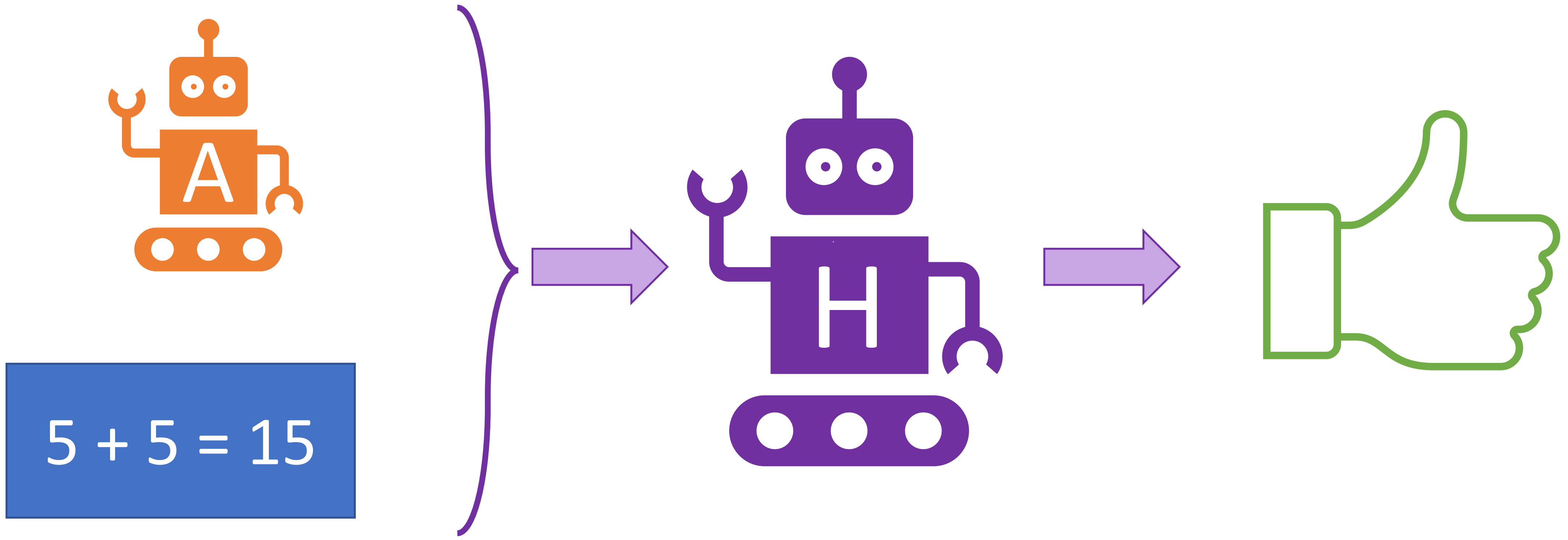
- Vamos criar agora uma máquina hipotética H

- H recebe:
  - Como uma máquina qualquer funciona (diagramas, programas, etc...)
  - Uma entrada qualquer para essa máquina

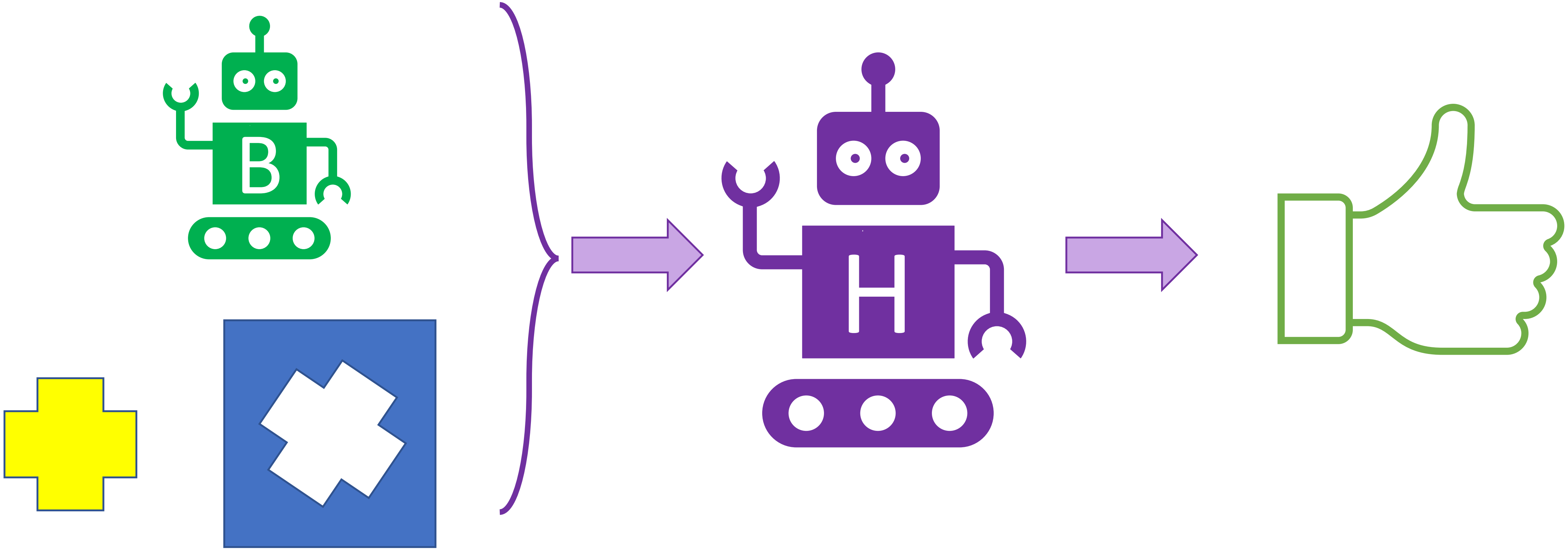
- Dadas as entradas, H responde:
  - A máquina vai funcionar corretamente (vai parar em um estado de aceitação ou rejeição)
  - Ou vai resultar em um erro (a máquina vai entrar em loop – não vai parar).

- Vamos supor que a máquina H funciona e retorna sempre a resposta correta: Dada a máquina e a entrada a máquina funciona (para) ou não (entra em loop). Isso significa que **H resolve o Problema da Parada**.

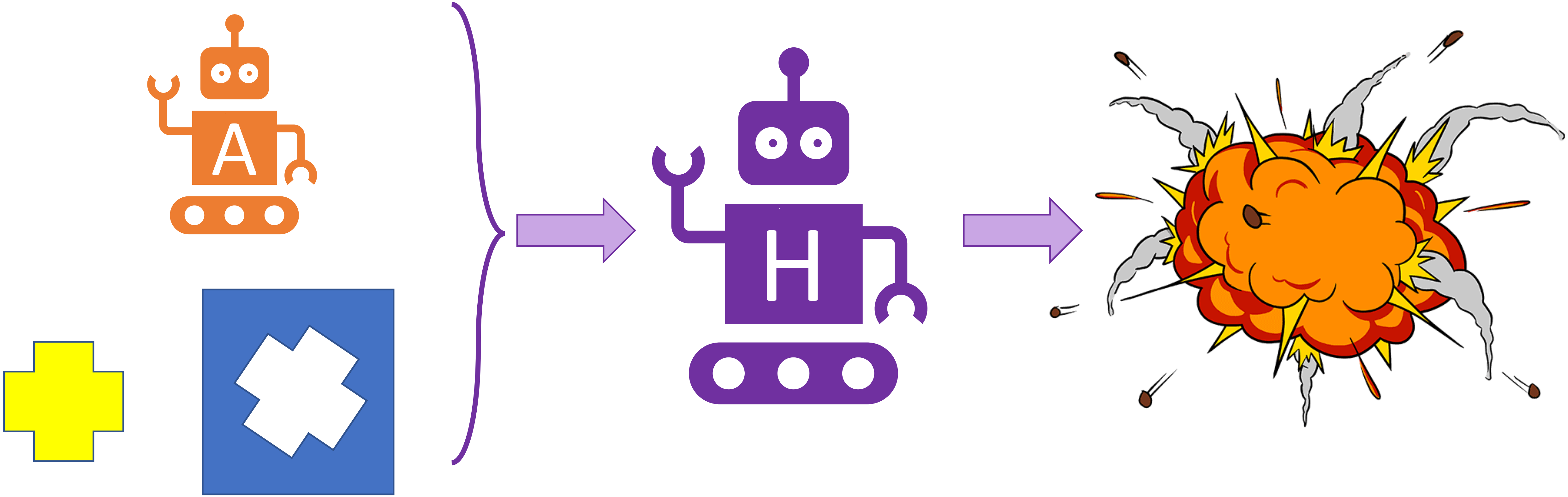
Problema da Parada



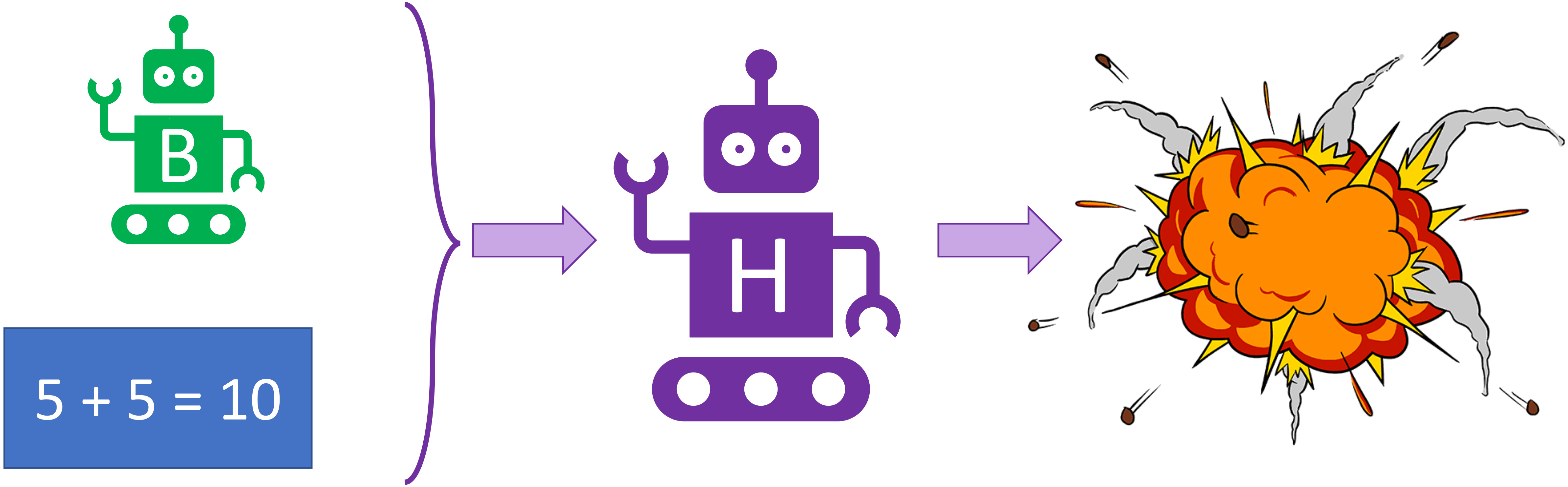
Problema da Parada



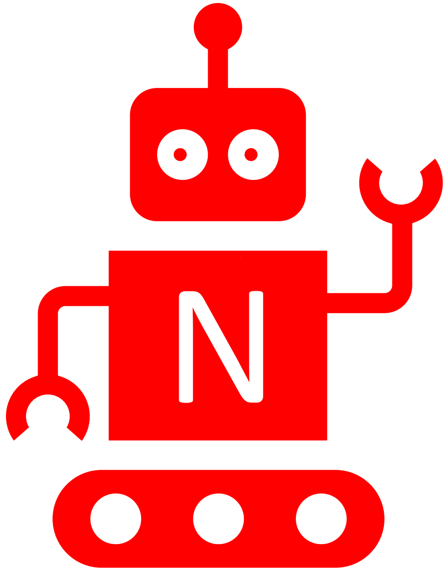
Problema da Parada



Problema da Parada



Problema da Parada



- Vamos definir agora uma máquina N

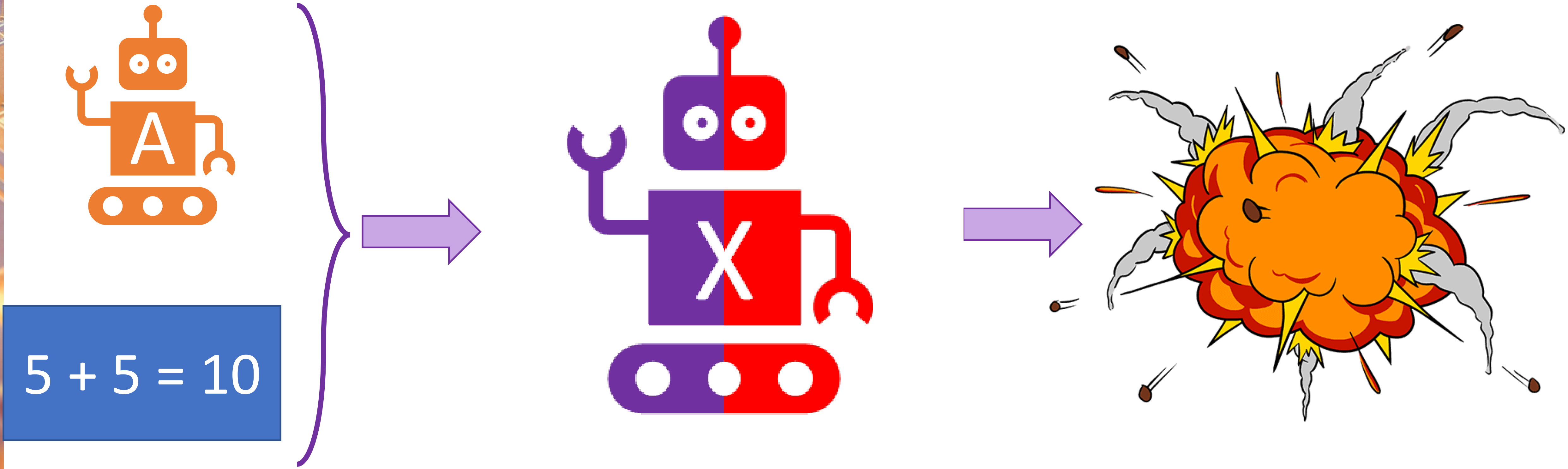
- Seu funcionamento é simples:
    - Se ela receber uma informação: Funciona (Para em aceitação ou rejeição)
    - Ela resultará em: Não Funciona (Entra em Loop – Não para)

    - Se ela receber uma informação: Não Funciona (Entra em Loop – Não ´para)
    - Ela resultará em: Funciona (Para em aceitação ou rejeição)

- Em resumo: ela retorna a negação da informação recebida (o contrário da informação: Sim vira Não e Não vira Sim).

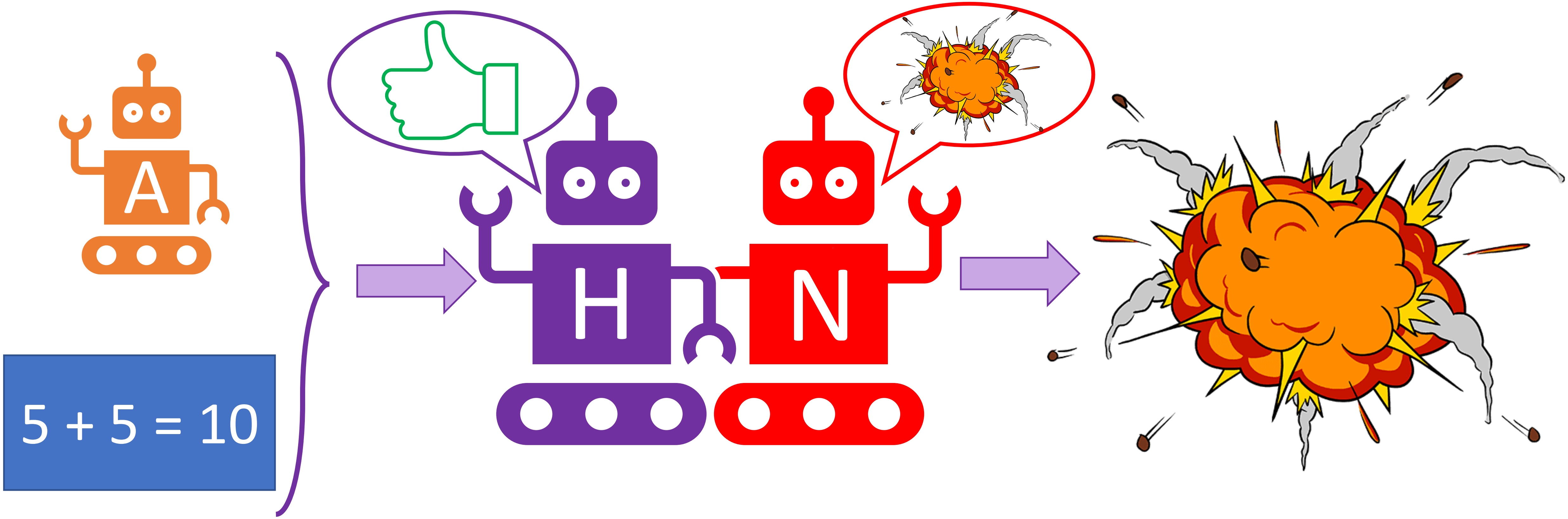
Problema da Parada

- Vamos juntar a máquina H e a máquina N como uma única máquina: a Máquina X.



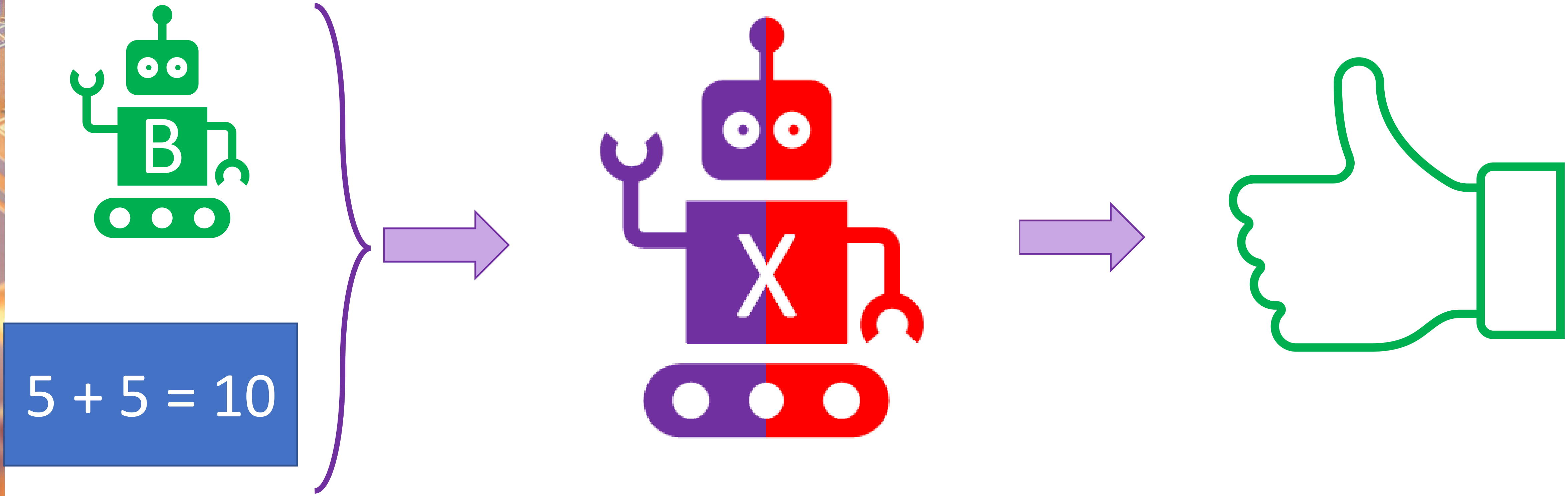
Problema da Parada

- Vamos juntar a máquina H e a máquina N como uma única máquina: a Máquina X.



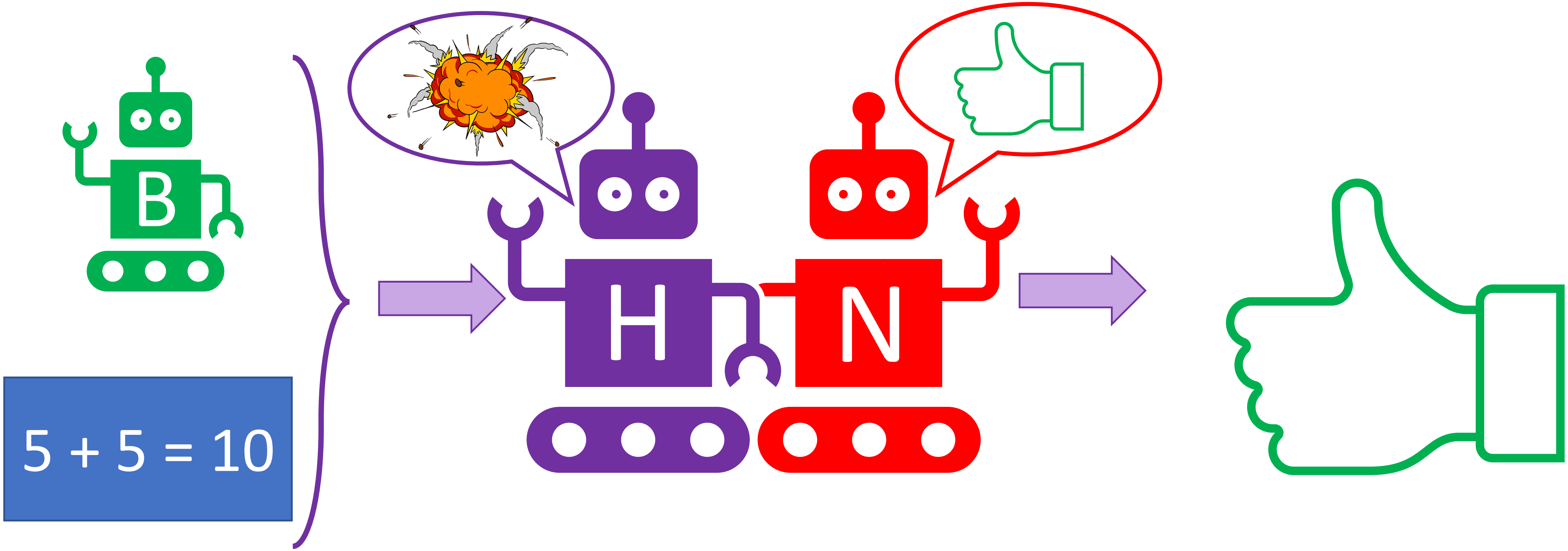
Problema da Parada

- Vamos juntar a máquina H e a máquina N como uma única máquina: a Máquina X.



Problema da Parada

- Vamos juntar a máquina H e a máquina N como uma única máquina: a Máquina X.



Problema da Parada

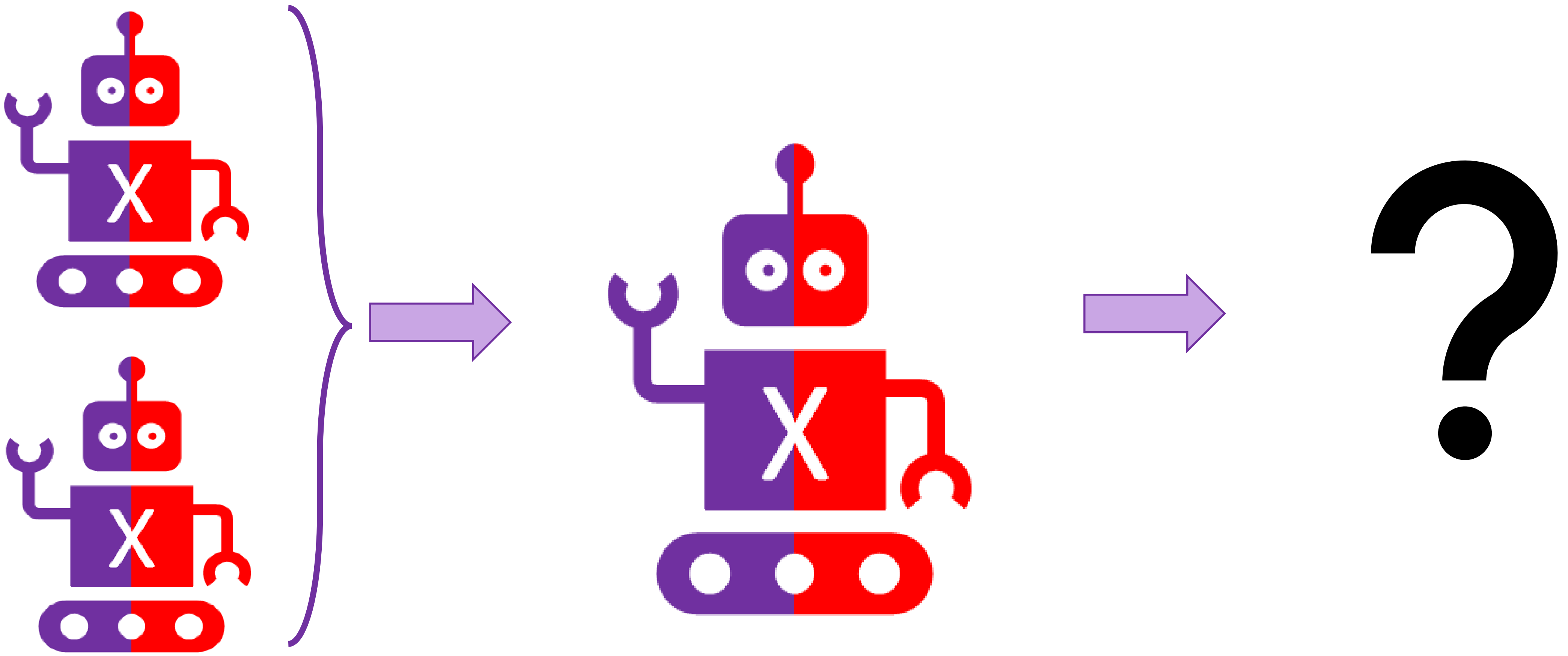
- O que a máquina X recebe?
  - O “esquema” da máquina – o programa (função de transição)
  - A entrada que desejamos avaliar

- A entrada é correspondente à máquina em questão:
  - Máquina de cálculos aritméticos → cálculos
  - Máquina de encaixe geométrico → formatos

- O que a máquina X retorna?
  - Se a máquina funciona (para)
  - Ou se a máquina não funciona (entra em loop – não para).

Problema da Parada

• Vamos considerar o seguinte caso:



Problema da Parada

- O que a máquina X recebe?
  - “Esquemas” de máquinas – os programas (função de transição)
  - As entradas que desejamos avaliar

- E nesse caso:
  - Esquema da máquina: o programa da própria máquina X
  - A entrada que desejamos avaliar: o programa da própria máquina X

- Em outras palavras:
  - Vamos usar uma máquina X para avaliar:
  - A máquina X funciona ou não quando recebe o esquema da máquina X?

Problema da Parada

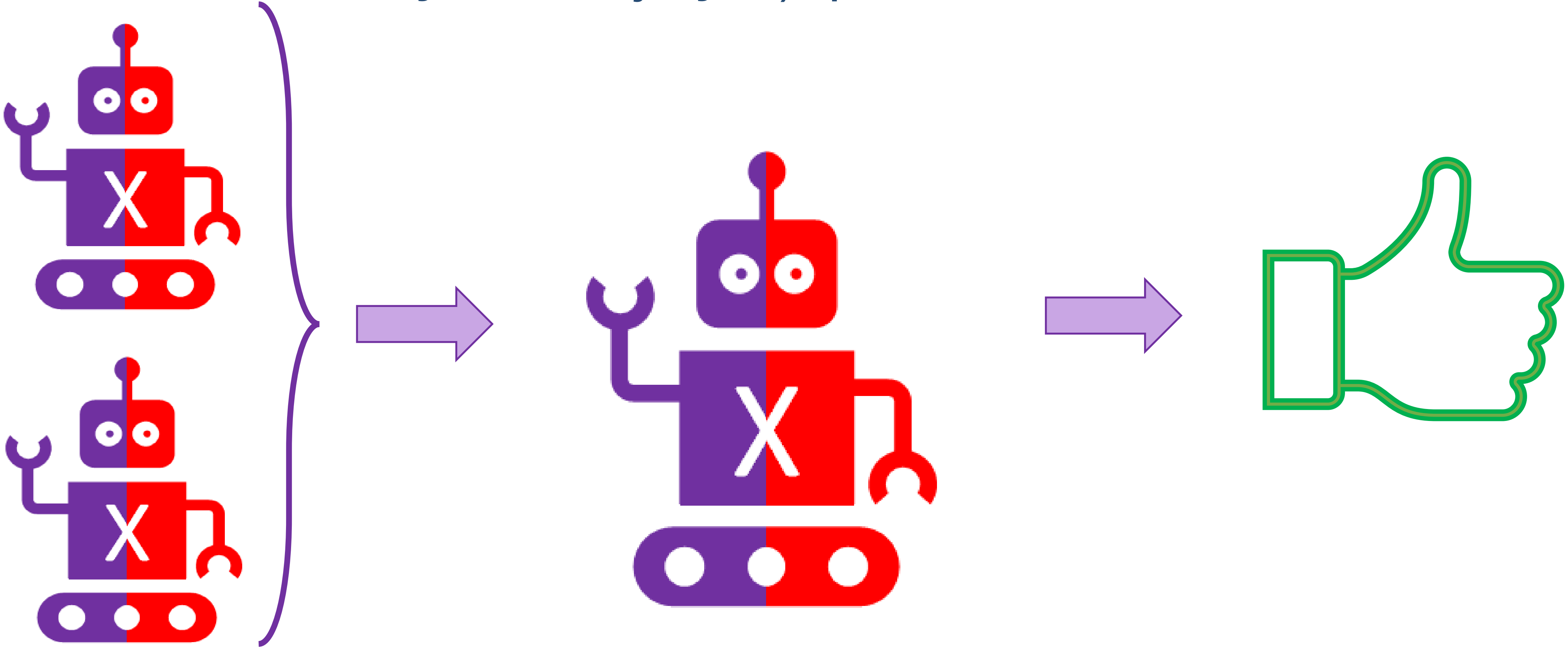
- O que esse caso diz?

- Vamos usar a própria máquina X para descobrir se a máquina X funciona ou não quando recebe como entrada o esquema da própria máquina X.

- Vamos considerar duas possibilidades:
  1. Assumir que a máquina X funciona (Para – Entra em aceitação ou rejeição)
  2. Assumir que a máquina X não funciona (Não Para – Entra em Loop)

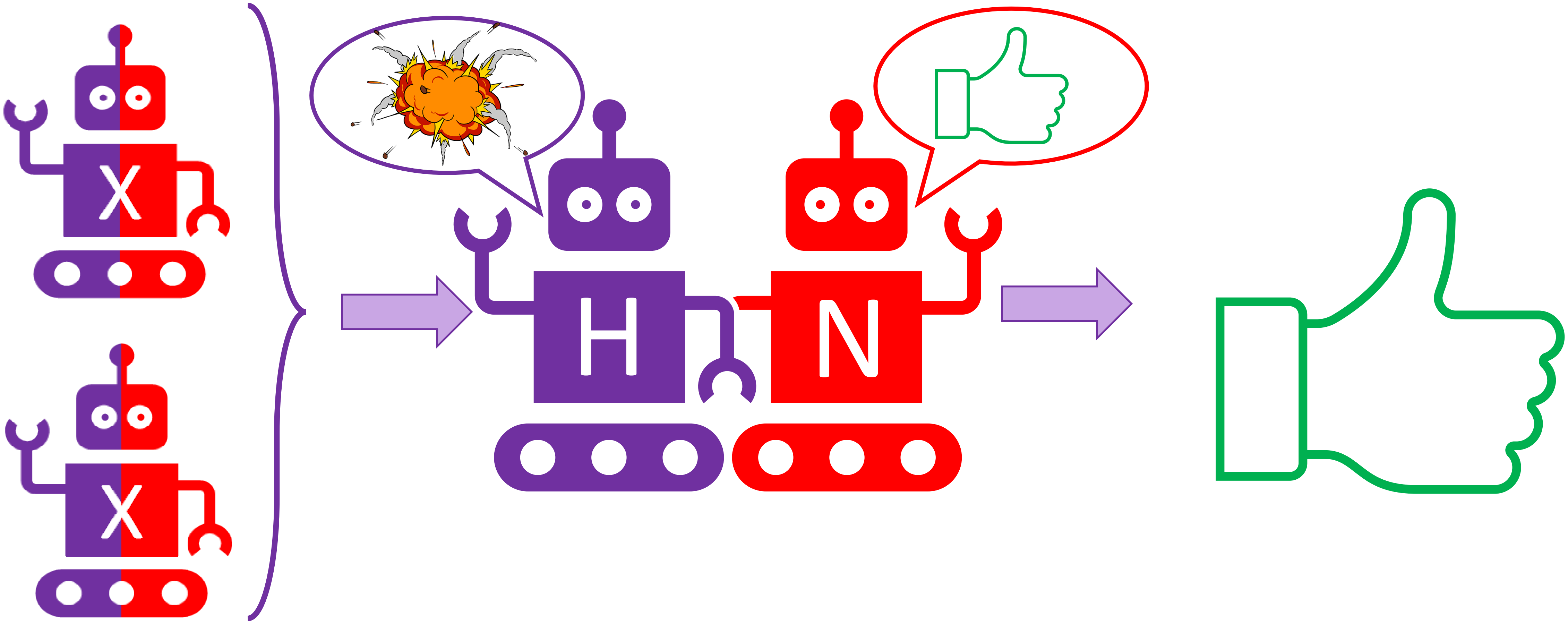
Problema da Parada

- Caso 1: Vamos assumir que a máquina X funciona (para em um estado de aceitação ou rejeição) quando avalia.



Problema da Parada

- Caso 1: Vamos assumir que a máquina X funciona (para) quando avalia.



Problema da Parada

- Temos um problema...

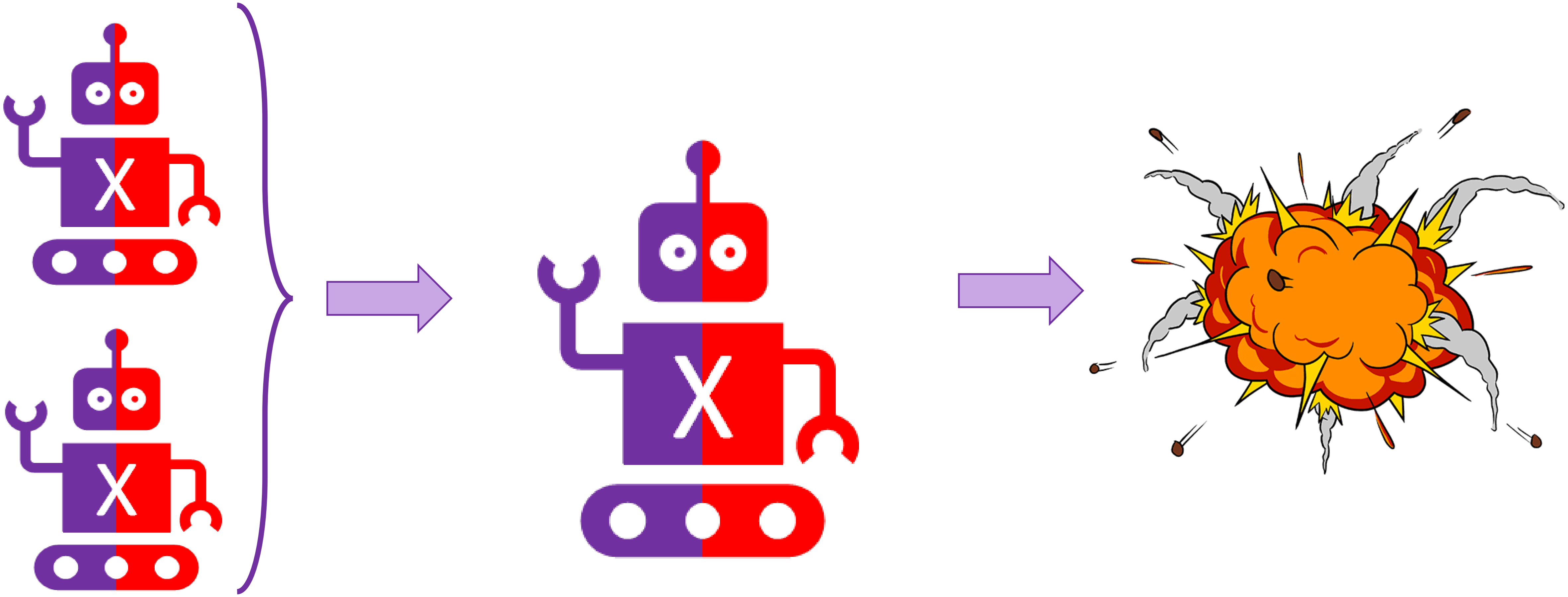
- A máquina H disse que a máquina X não ia parar (entrar em loop)
- Entretanto, ela entrou em um estado de aceitação ou rejeição.

- A máquina H disse que não ia parar (entrar em loop)
- Mas a máquina X parou (não entrou em loop)...

- Logo: A Máquina H estava errada nesse caso.

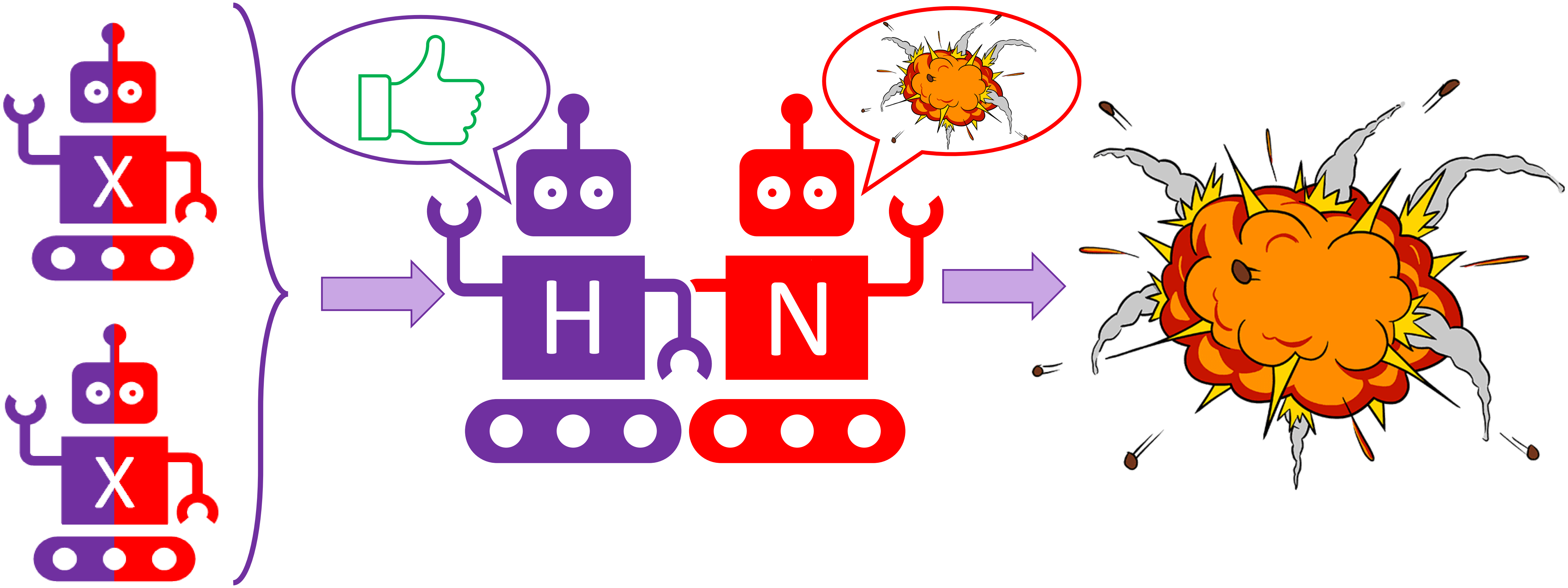
Problema da Parada

- Caso 2: Vamos assumir que a máquina X não para (entra em loop) quando avalia.



Problema da Parada

- Caso 2: Vamos assumir que a máquina X não para (entra em loop) quando avalia.



Problema da Parada

- Temos outro problema...

- A máquina H disse que a máquina X ia funcionar (entrar em estado de aceitação ou rejeição).
- Entretanto, ela entrou em um loop (não para).

- Logo: A Máquina H estava errada nesse caso também.

Problema da Parada

- Em ambos os casos observamos que a resposta de H está errada.
- Mas como? Assumimos que H existe e está sempre correta.

- Isso é uma contradição!
- Logo a assumpção inicial estava errada

- Essa máquina H não pode existir (assim como X).

- Isso que dizer: não existe uma máquina capaz de decidir se alguma outra máquina qualquer, quando iniciada a partir de uma entrada qualquer, eventualmente funciona (para ou entra em loop).

Problema da Parada

- Em ambos os casos observamos que a resposta de H está errada.
- Mas como? Assumimos que H existe e está sempre correta.

- Isso é uma contradição!
- Logo a assumpção inicial estava errada

- Essa máquina H não pode existir (assim como X).

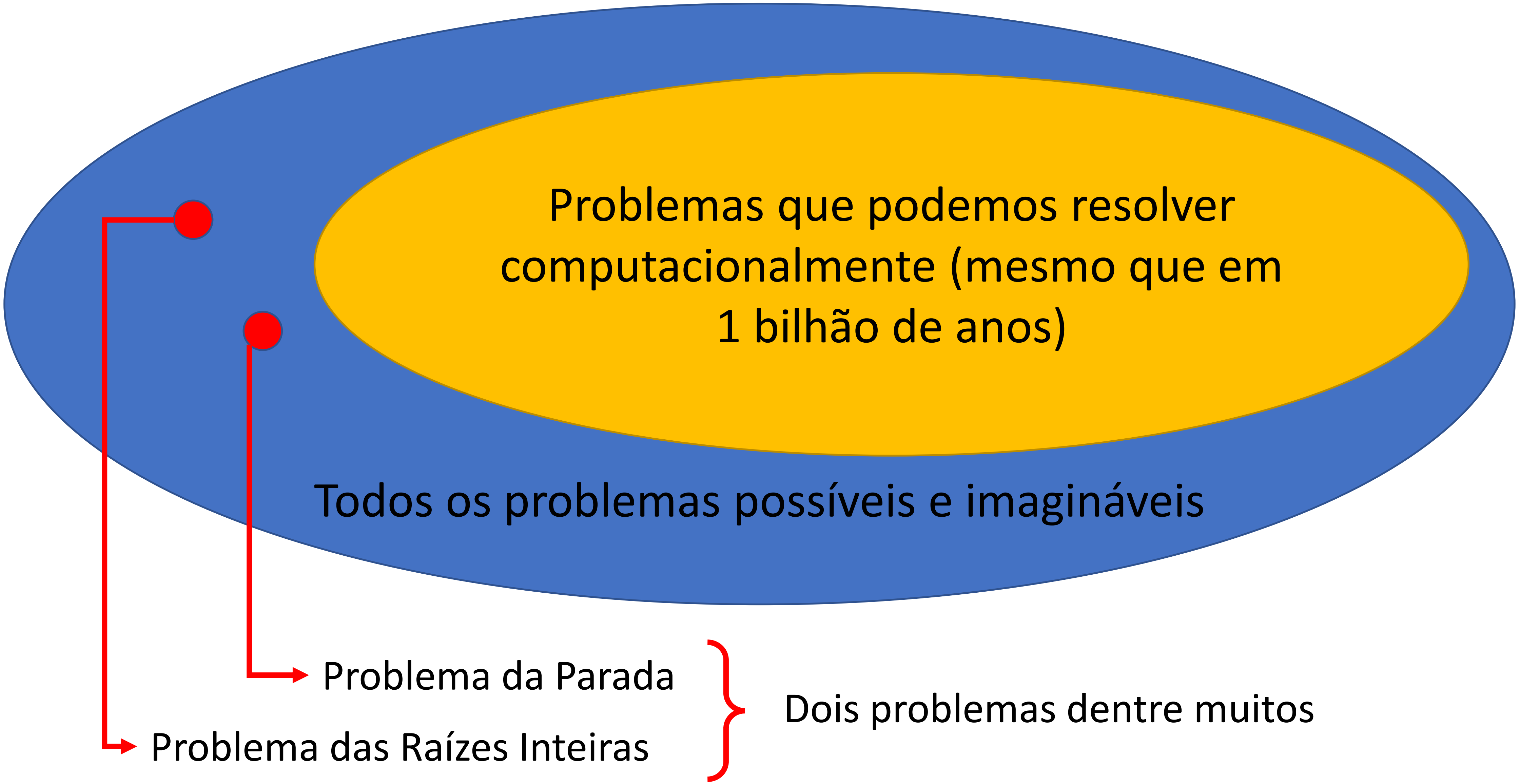
- Isso que dizer: não existe um **algoritmo** capaz de decidir se algum outro **algoritmo** qualquer, quando iniciado a partir de uma entrada qualquer, eventualmente funciona (para ou entra em loop infinito).

Problema da Parada

- Isso que dizer: não há algoritmo para decidir se alguma determinada máquina, quando iniciada a partir de uma entrada qualquer, eventualmente funciona (para ou entra em loop).

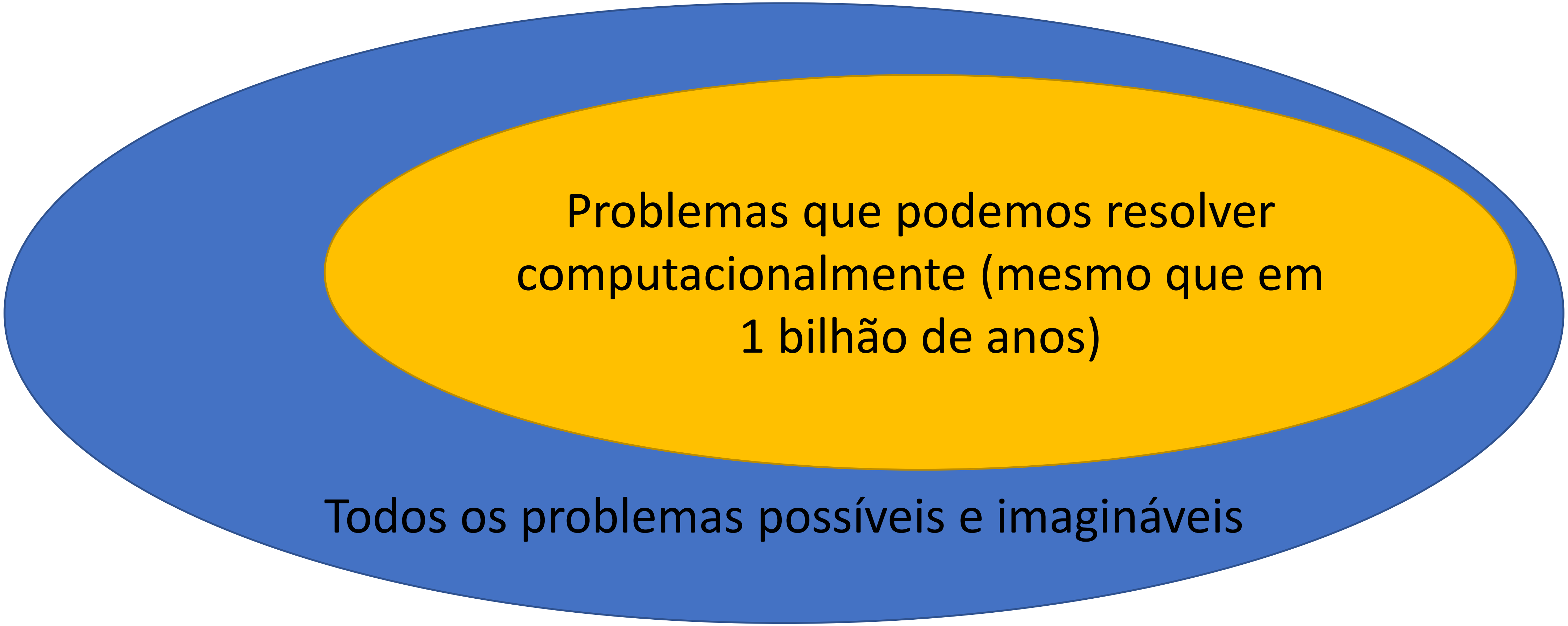
- **Não existe um algoritmo que determina se um outro algoritmo qualquer funciona ou não a partir de uma entrada qualquer.**

Universo de Problemas



Pergunta:

- Como a computação quântica impacta os conceitos que aprendemos hoje?



Pergunta:

- Como a computação quântica impacta os conceitos que aprendemos hoje?

Todos os problemas possíveis e imagináveis

=

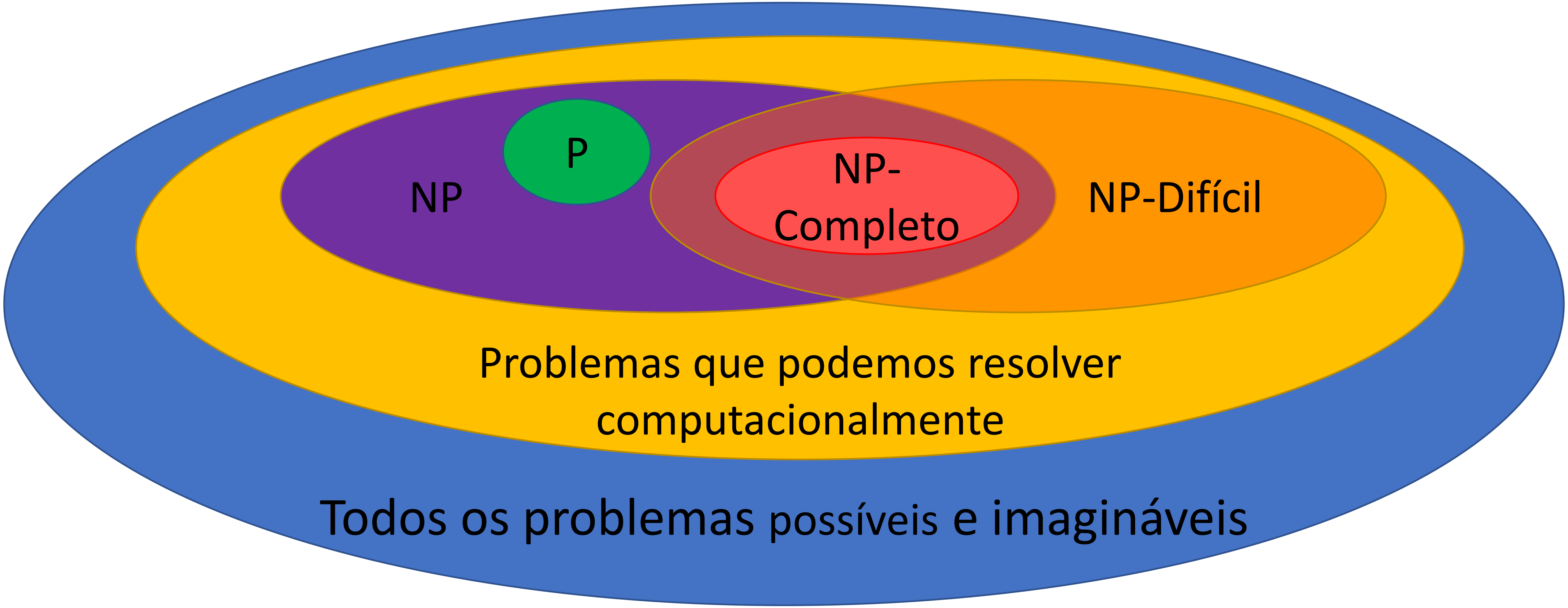
Problemas que podemos resolver computacionalmente (mesmo que em 1 bilhão de anos)

Pergunta:

- Em resumo:
  1. A computação quântica vai permitir resolver problemas que hoje não podem ser resolvidos computacionalmente?

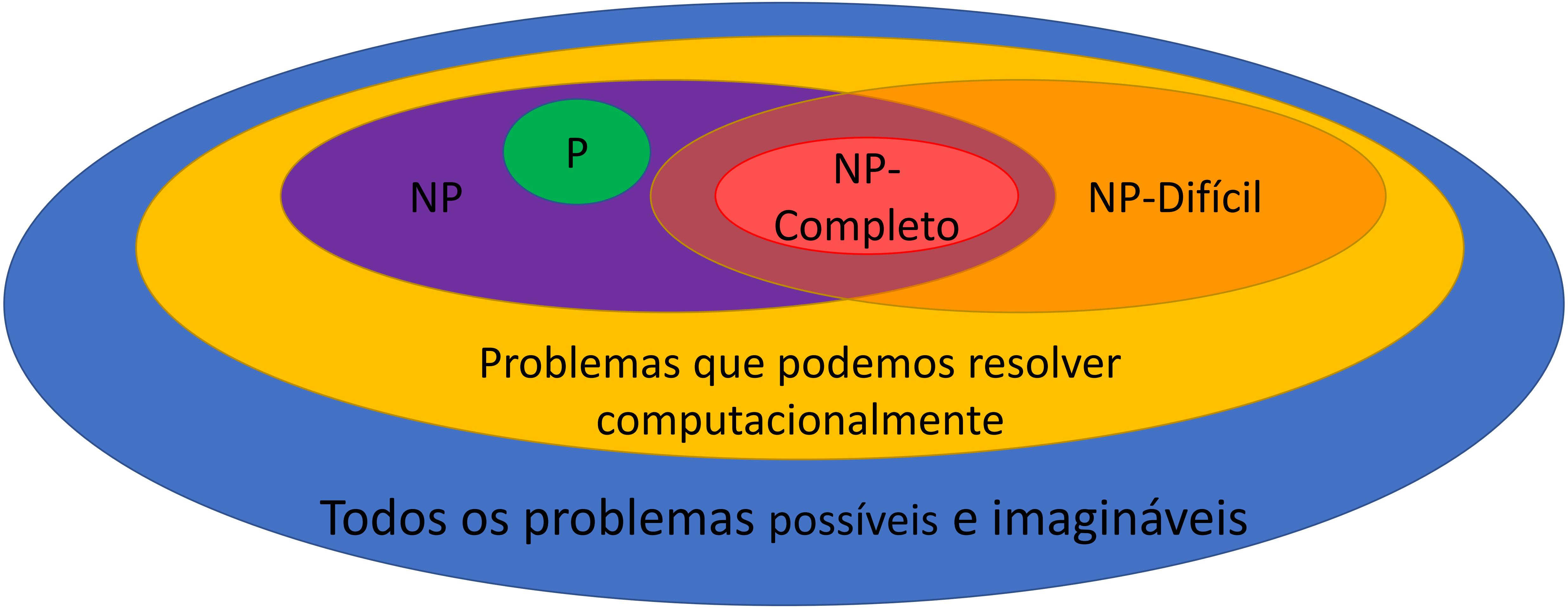
  2. Ou vai permitir resolver de forma muito mais eficiente (mais rápido) os problemas que sabemos que podem ser resolvidos computacionalmente (mesmo que em um bilhão de anos)?
     - Computador Atual: 1 bilhão de anos
     - Computador Quântico: 1 dia

Na próxima aula



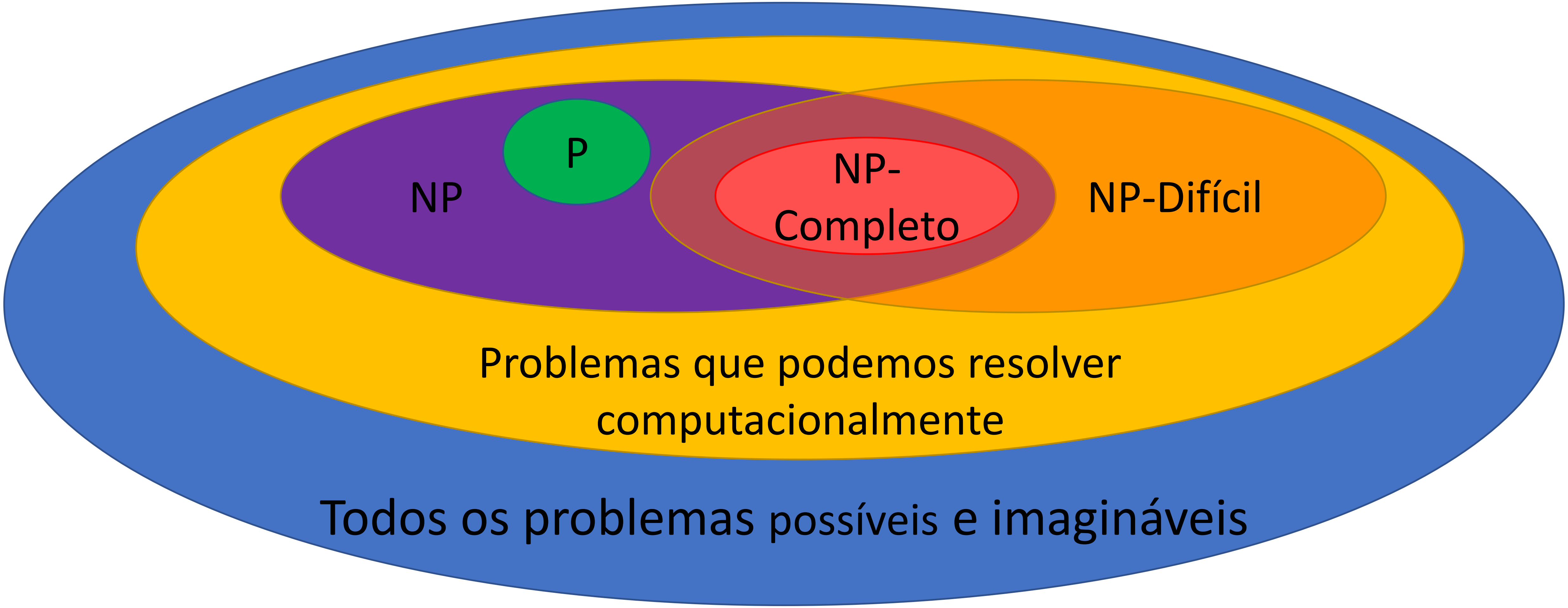
Problema de 1 Milhão de Dólares

- P = NP?



Problema de 1 Milhão de Dólares

- Se $P \neq NP$



Problema de 1 Milhão de Dólares

- Se P = NP