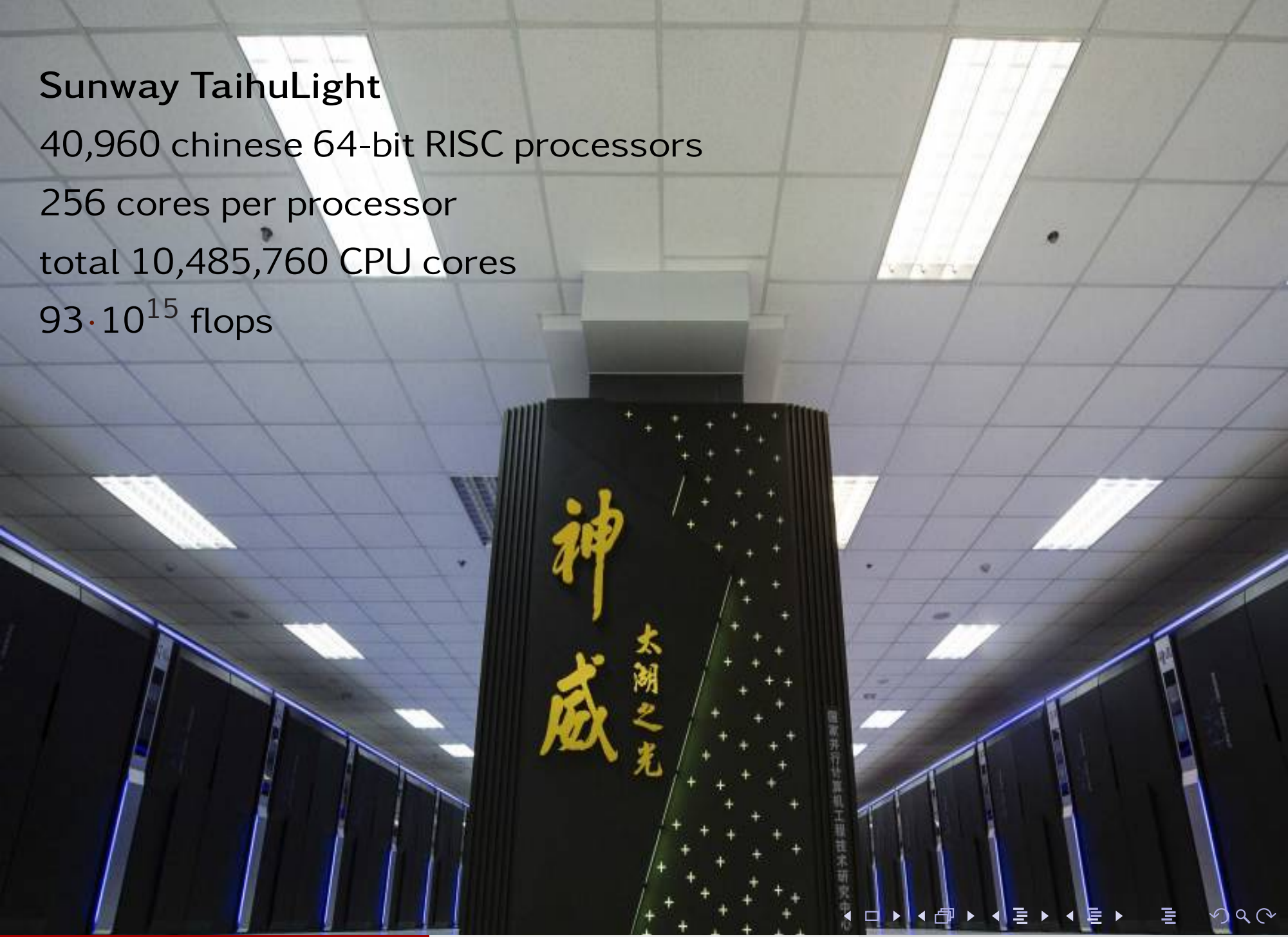
On Problems that Computers will Never Solve



Sunway TaihuLight

40,960 chinese 64-bit RISC processors

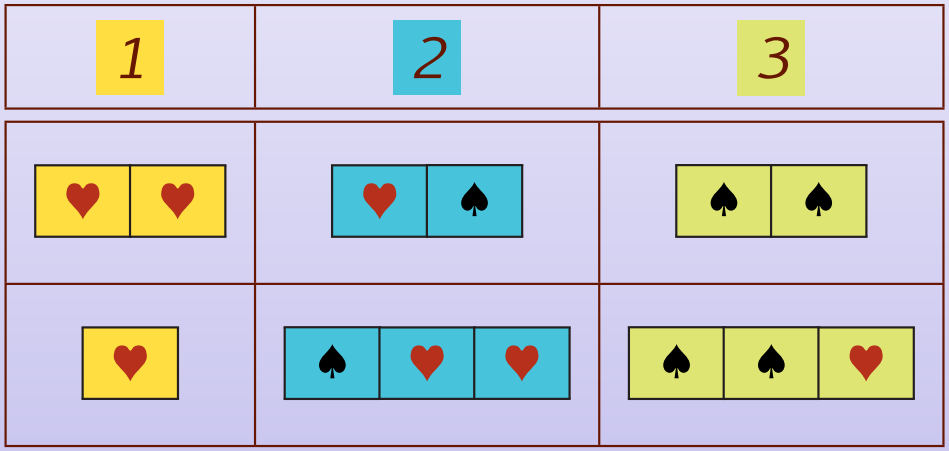
256 cores per processor

total 10,485,760 CPU cores

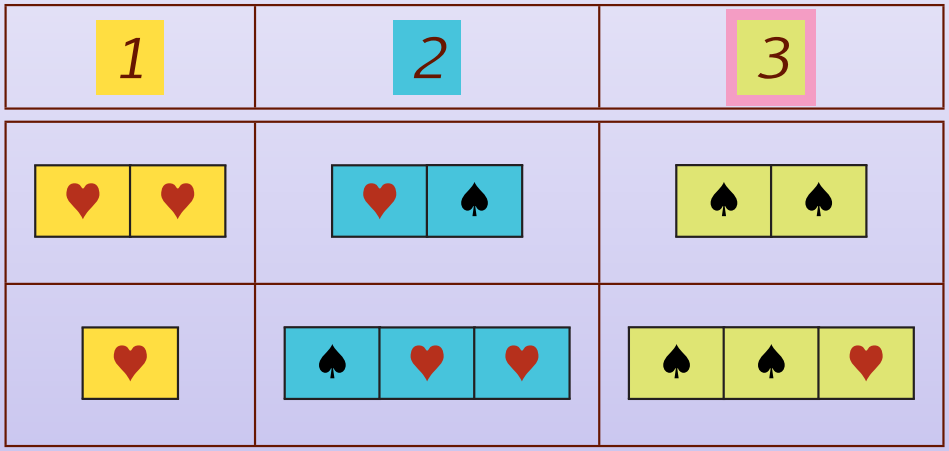
$93 \cdot 10^{15}$ flops

Let's try a puzzle

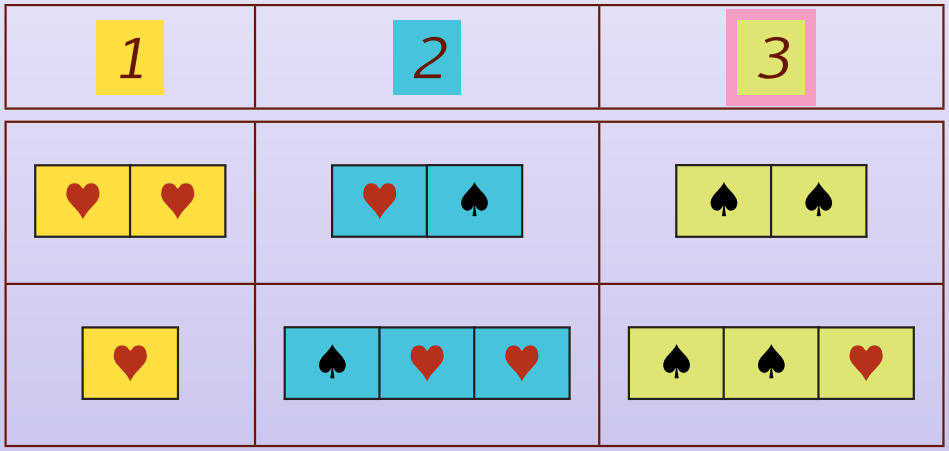
Let's try a puzzle



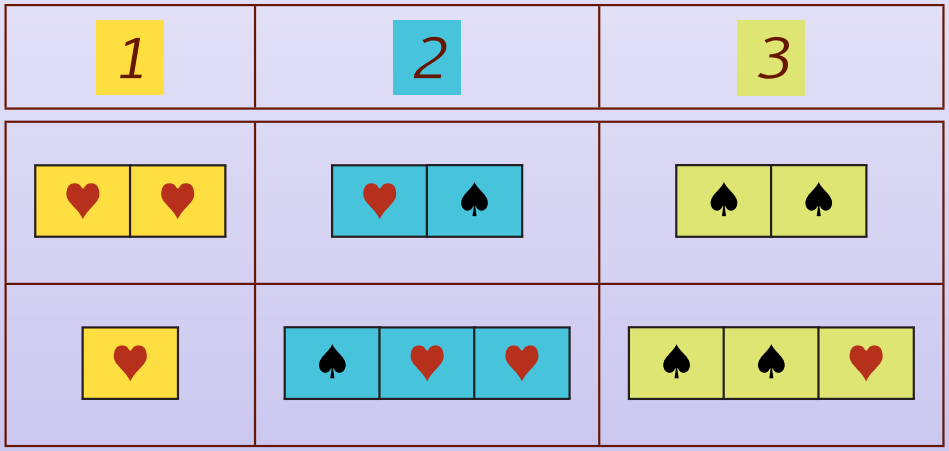
Let's try a puzzle



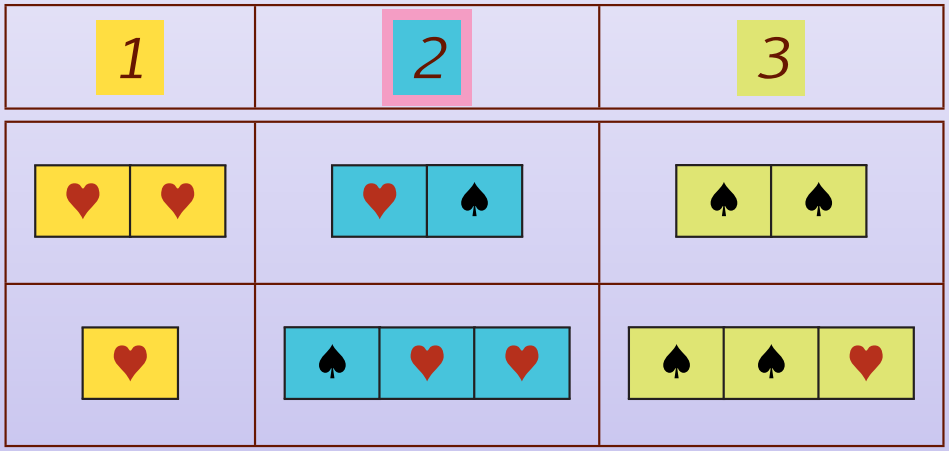
Let's try a puzzle



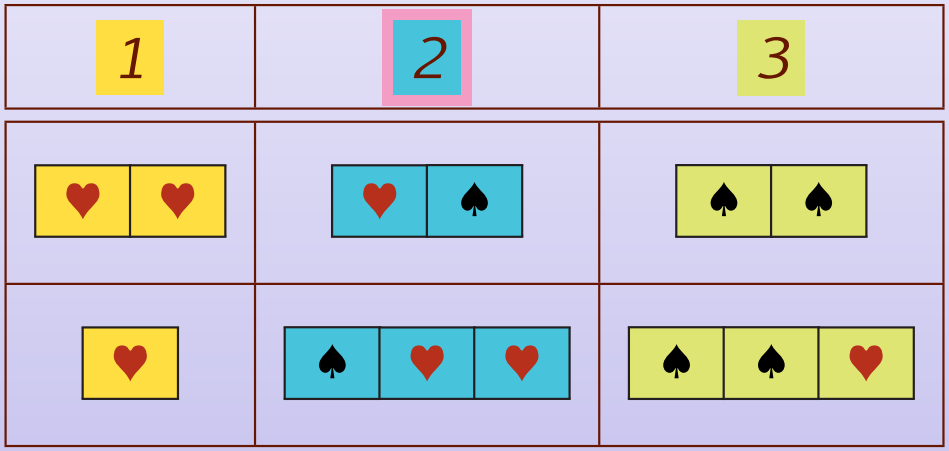
Let's try a puzzle



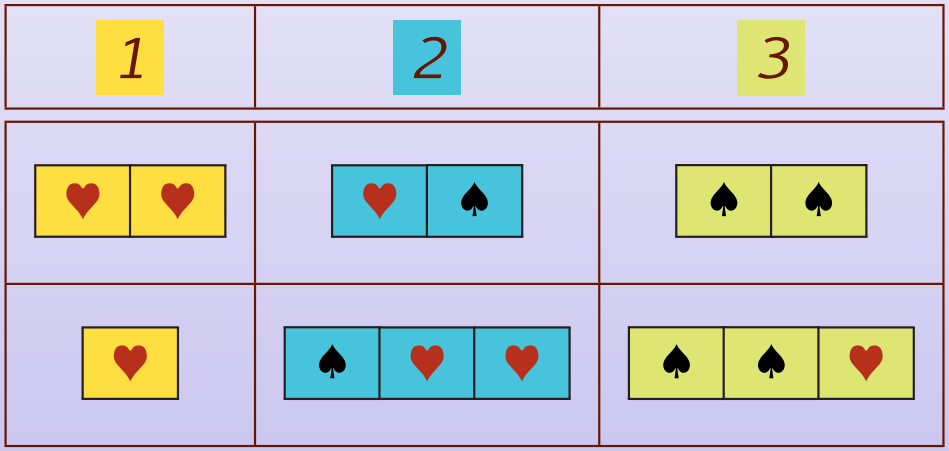
Let's try a puzzle



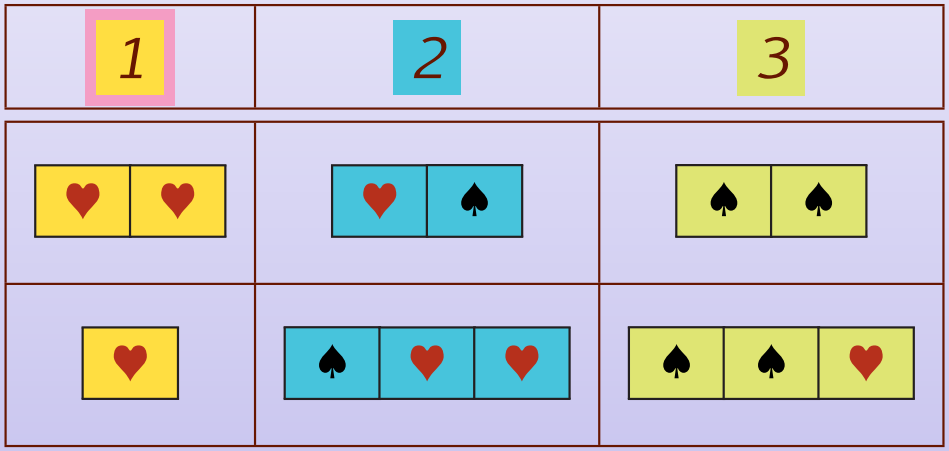
Let's try a puzzle



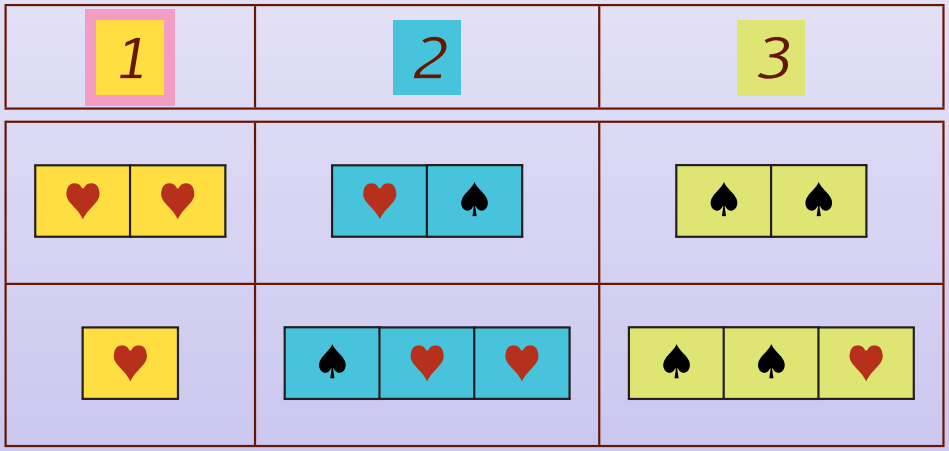
Let's try a puzzle

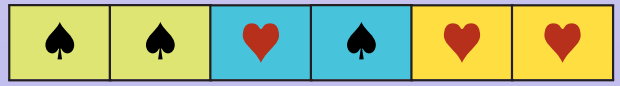


Let's try a puzzle

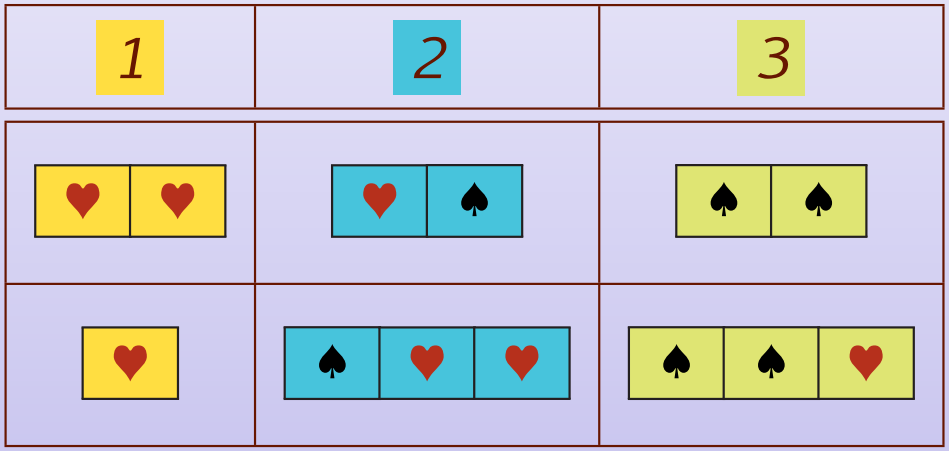


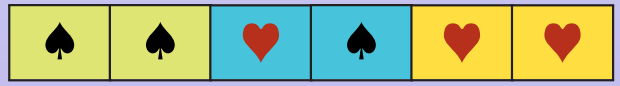
Let's try a puzzle



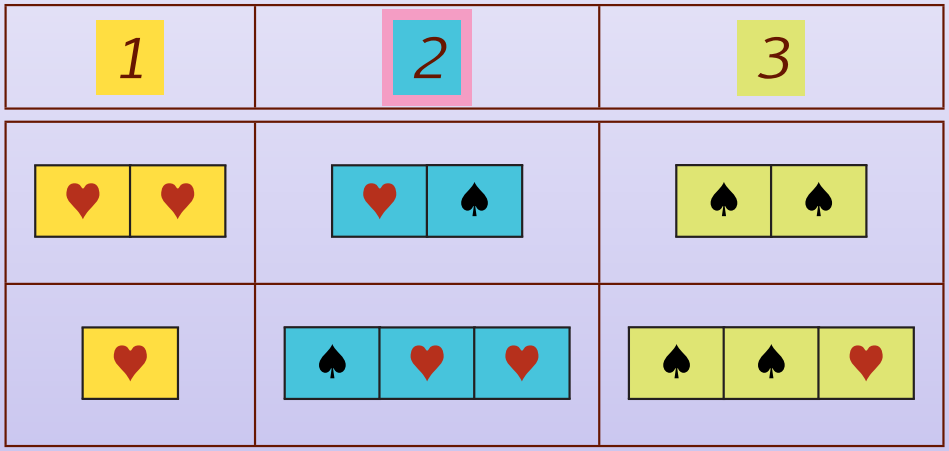


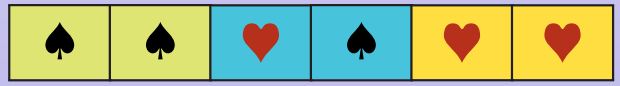
Let's try a puzzle



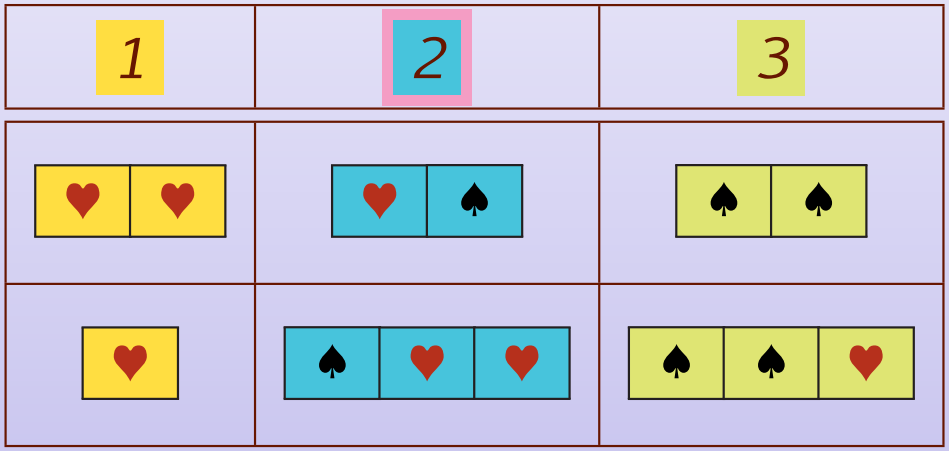


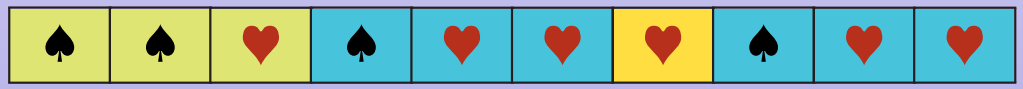
Let's try a puzzle



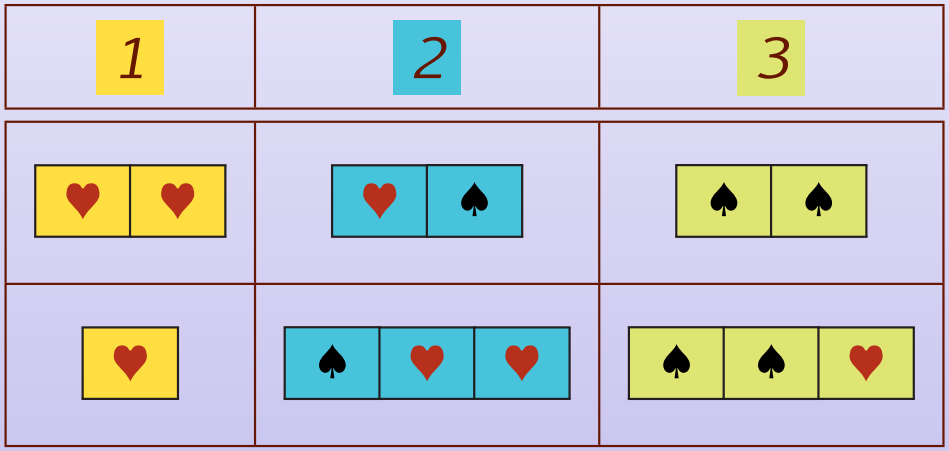


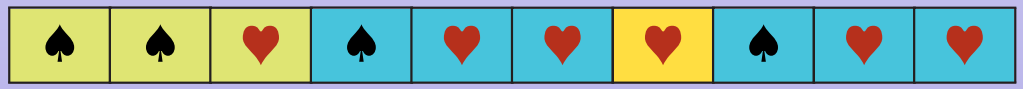
Let's try a puzzle



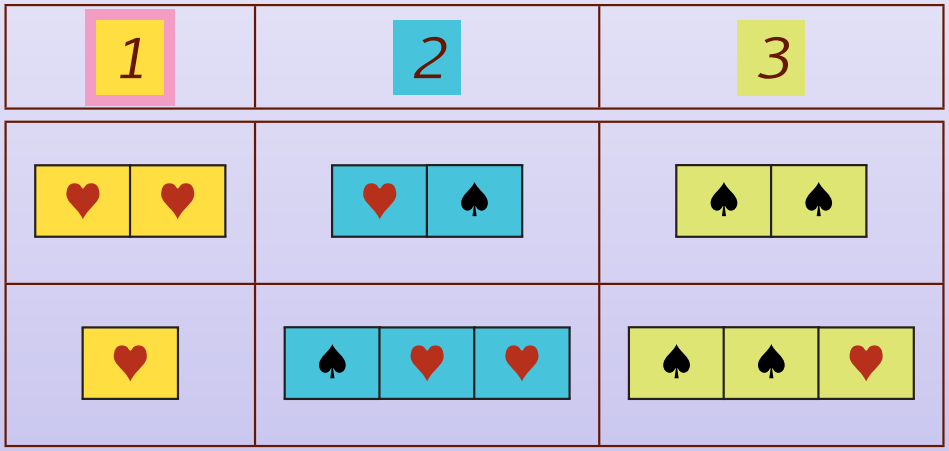


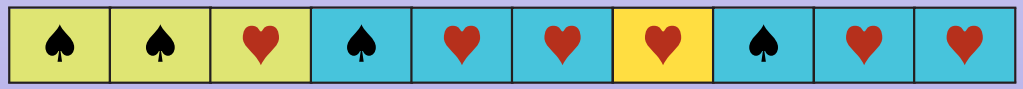
Let's try a puzzle



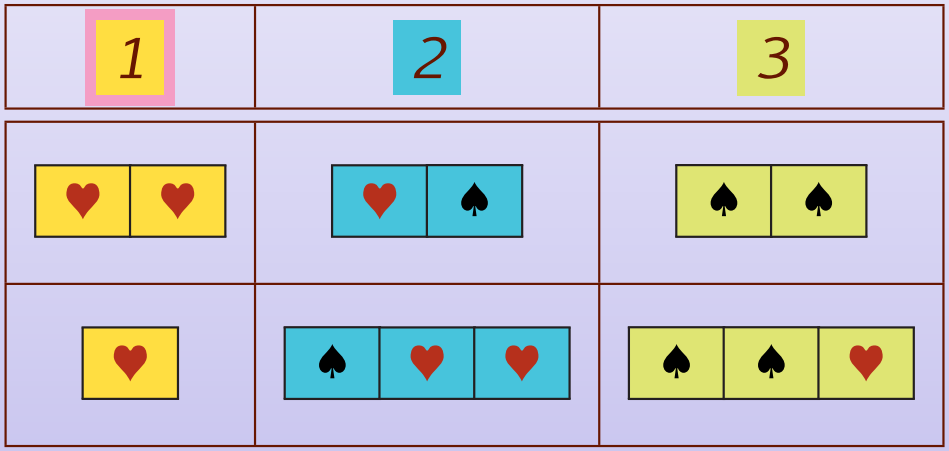


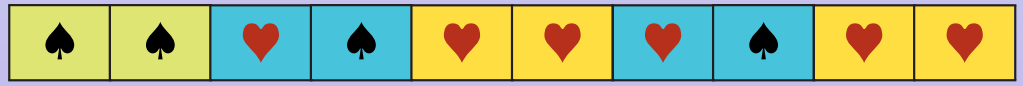
Let's try a puzzle



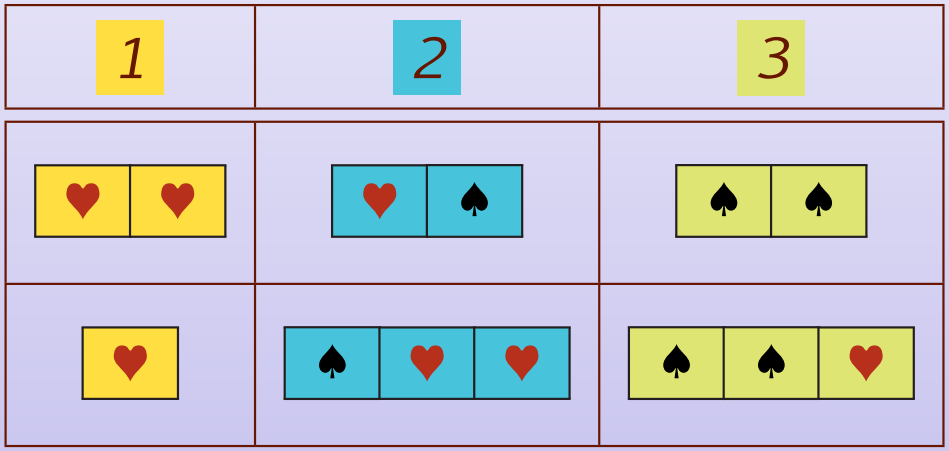


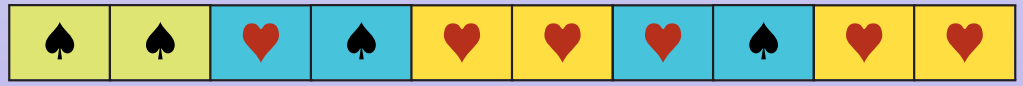
Let's try a puzzle



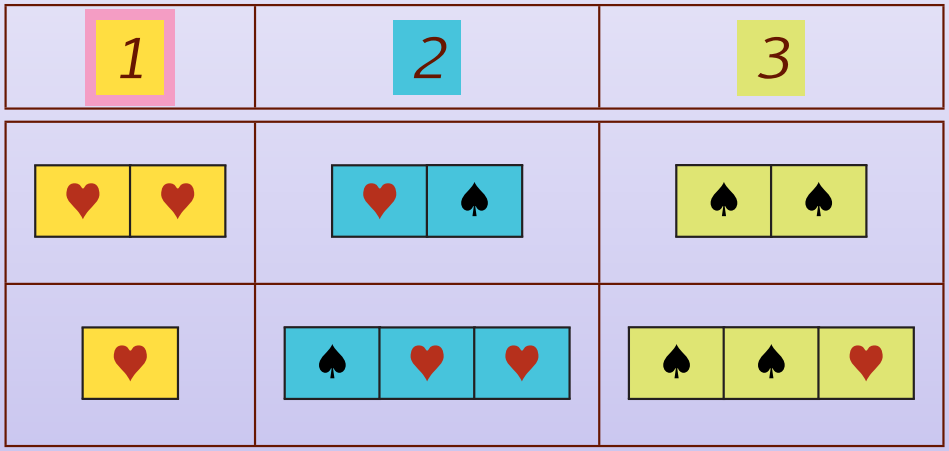


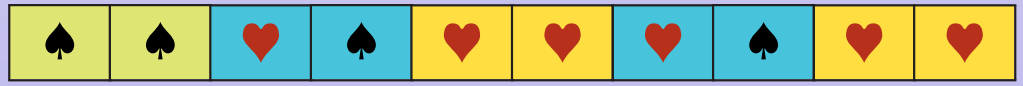
Let's try a puzzle



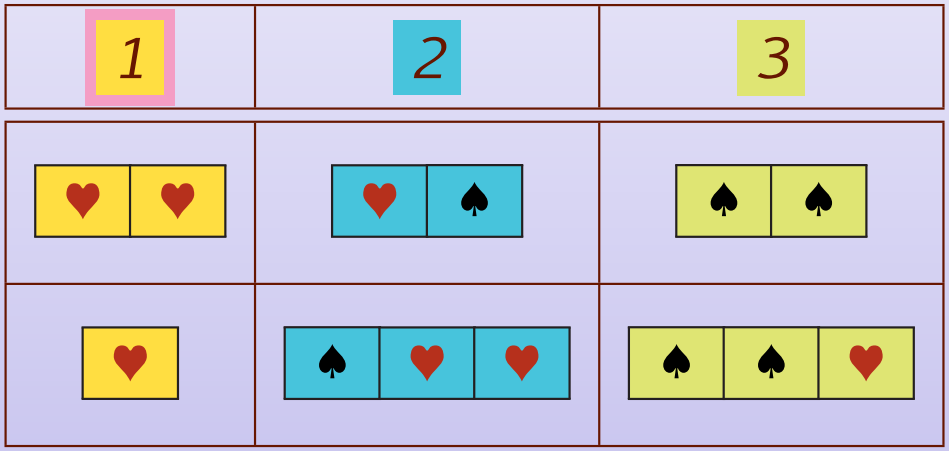


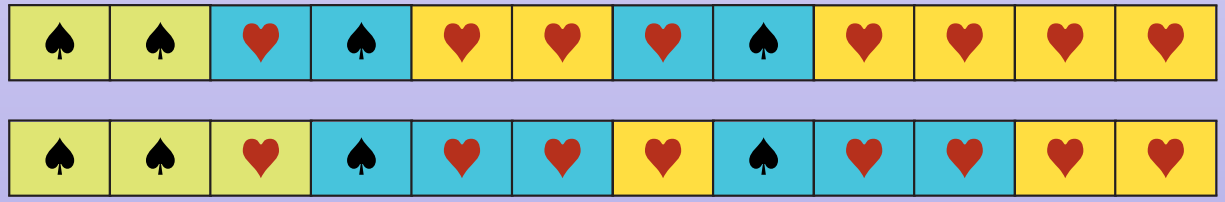
Let's try a puzzle



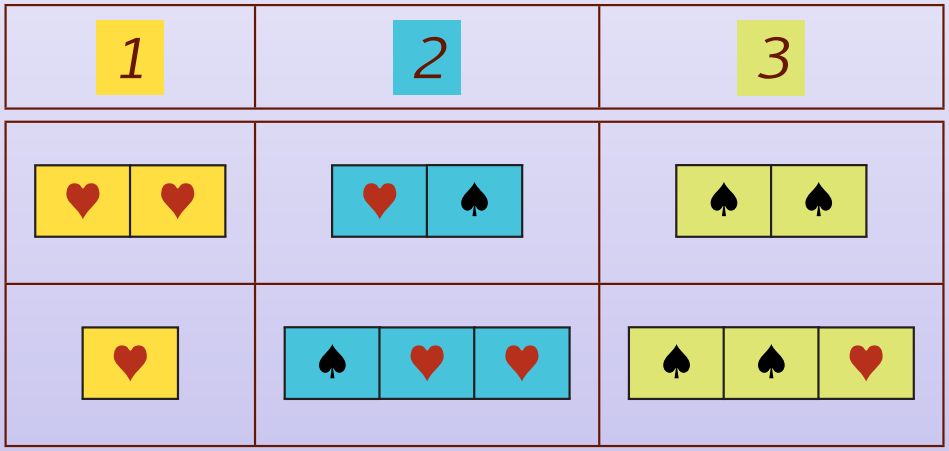


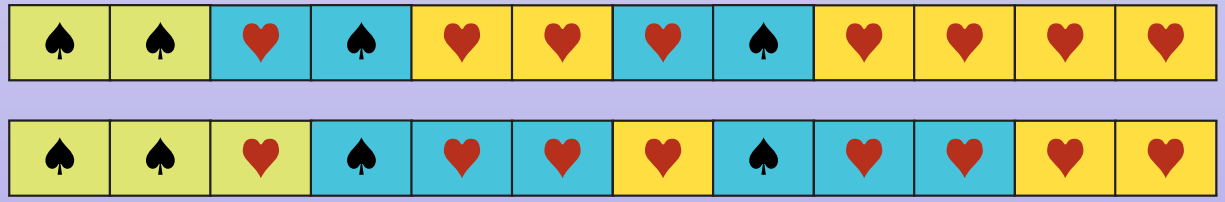
Let's try a puzzle





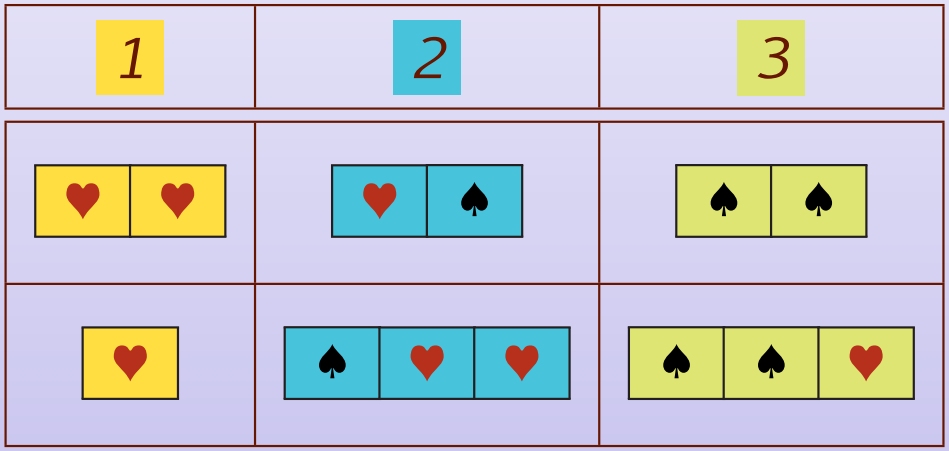
Let's try a puzzle





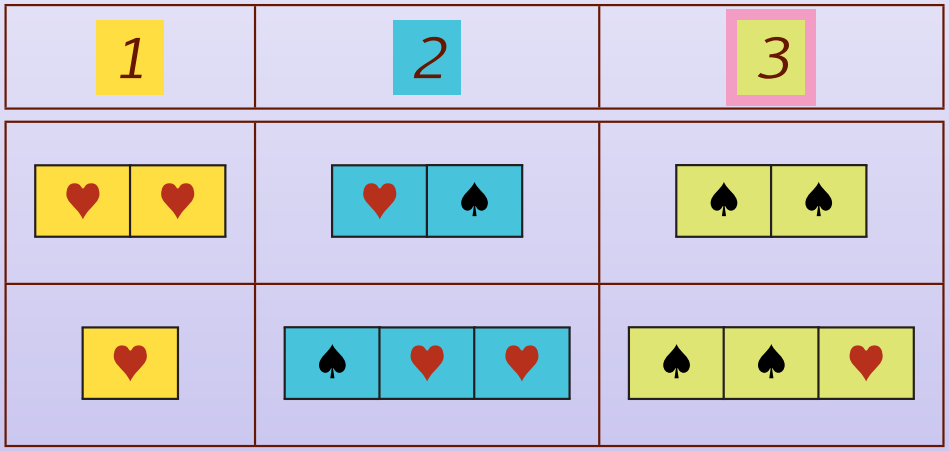
Done

Let's try a puzzle



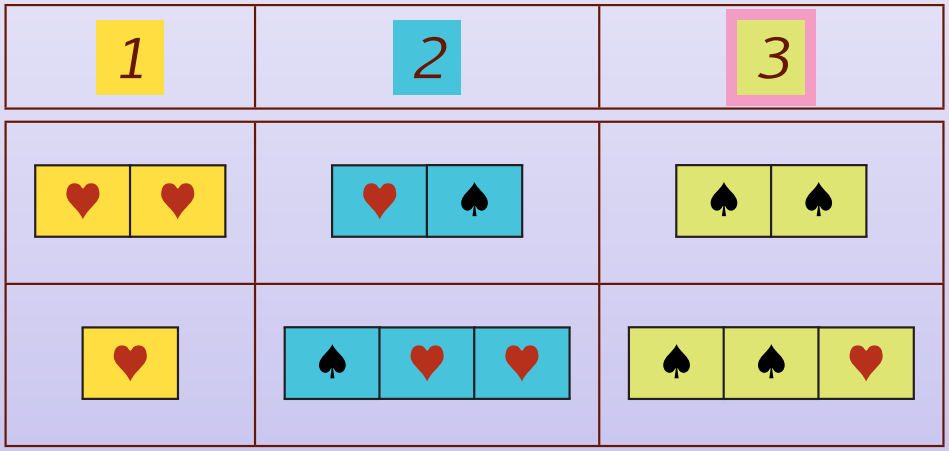
2nd play

Let's try a puzzle

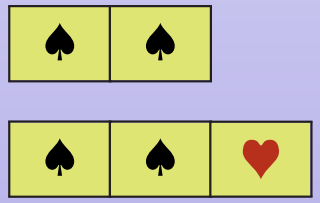


2nd play

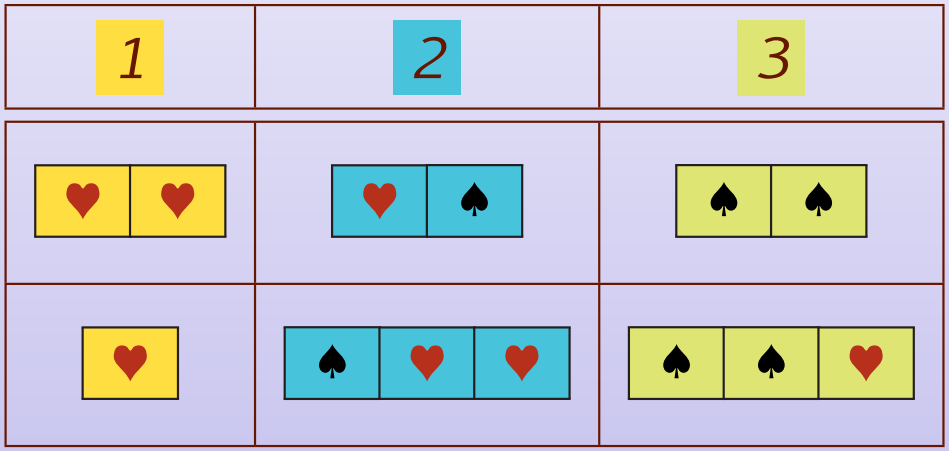
Let's try a puzzle



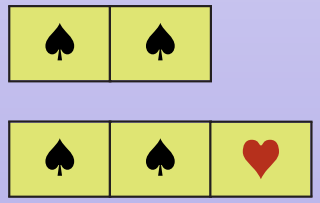
2nd play



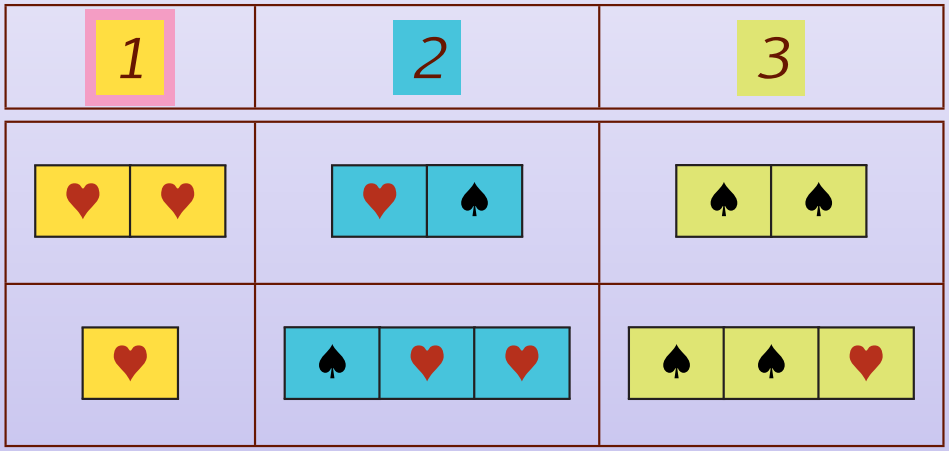
Let's try a puzzle



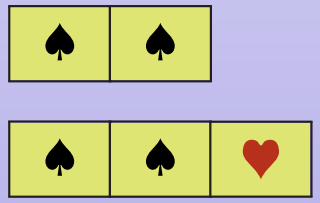
2nd play



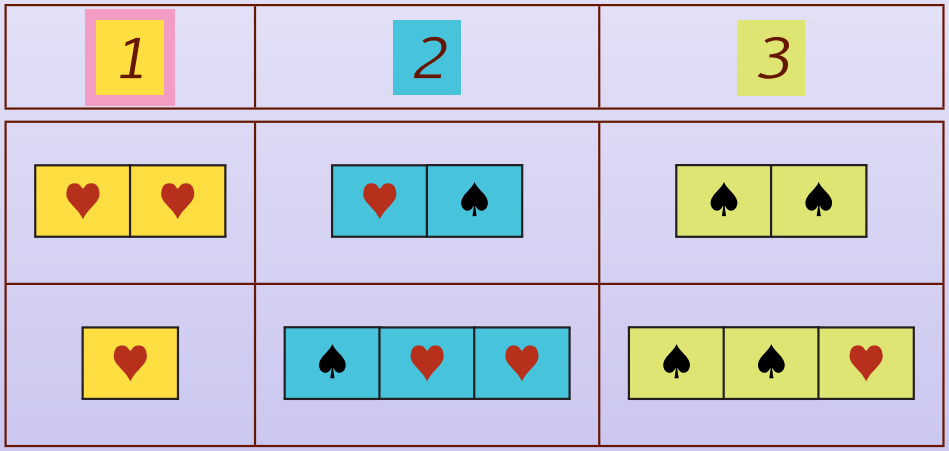
Let's try a puzzle



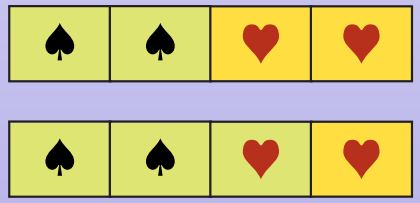
2nd play



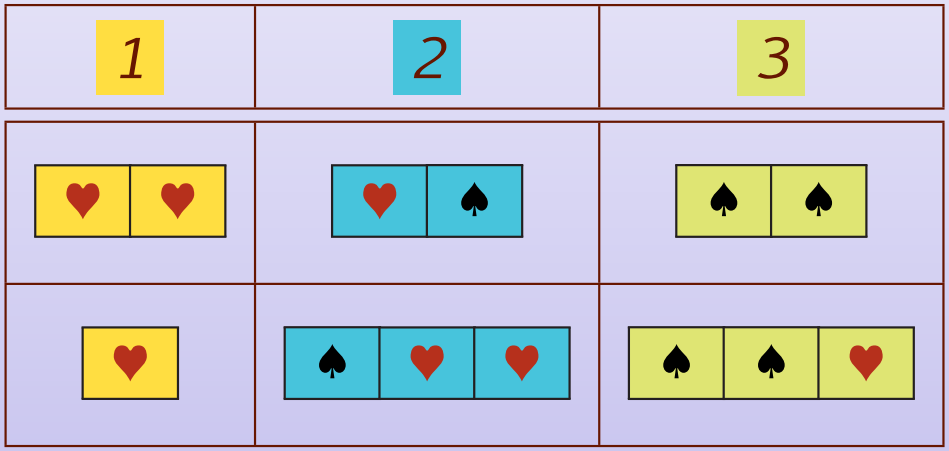
Let's try a puzzle



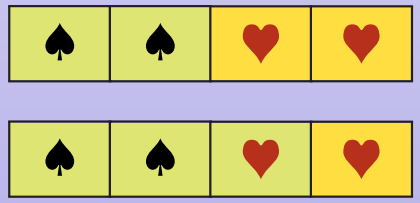
2nd play



Let's try a puzzle



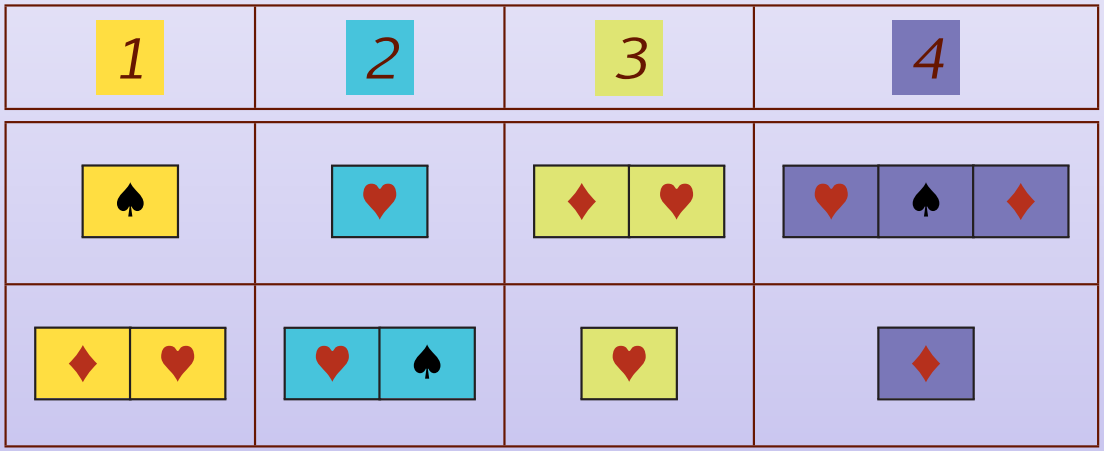
2nd play



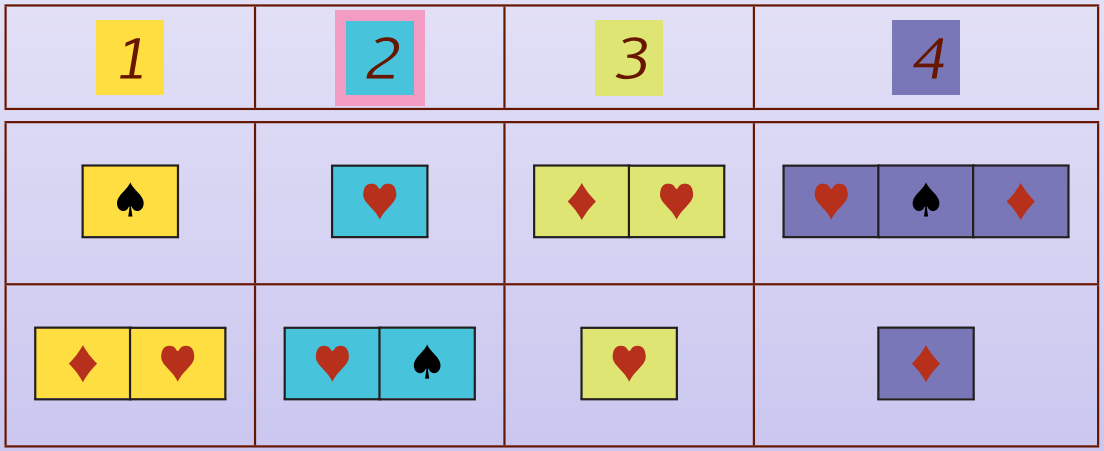
Done

Other instance of the same kind of puzzle

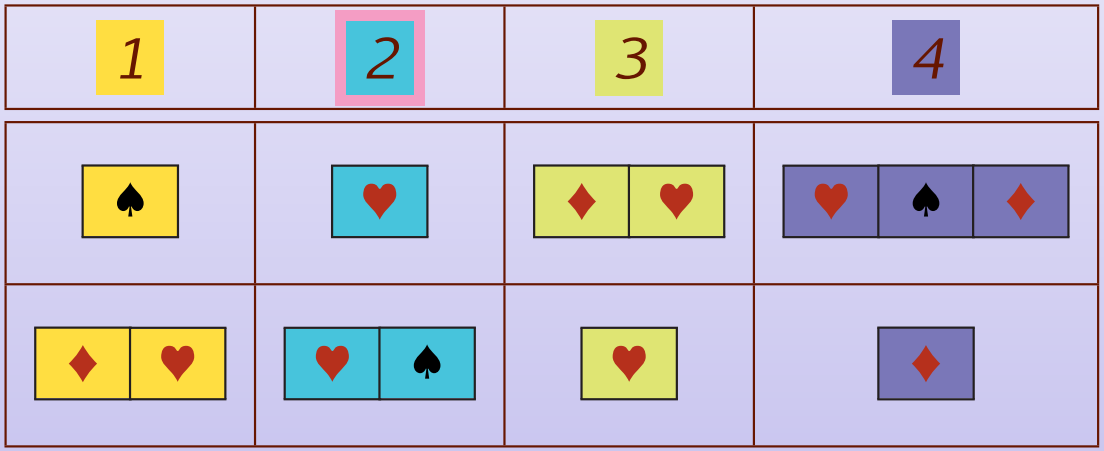
Other instance of the same kind of puzzle



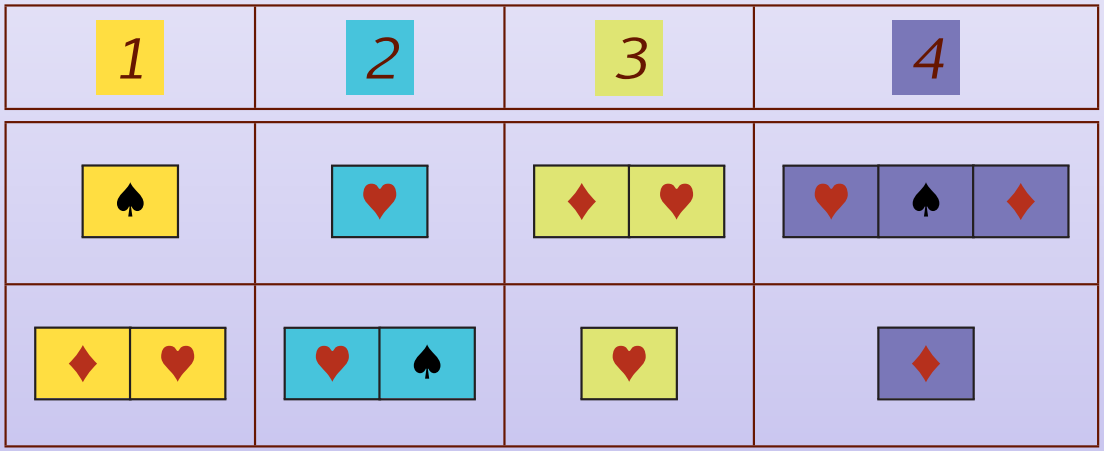
Other instance of the same kind of puzzle



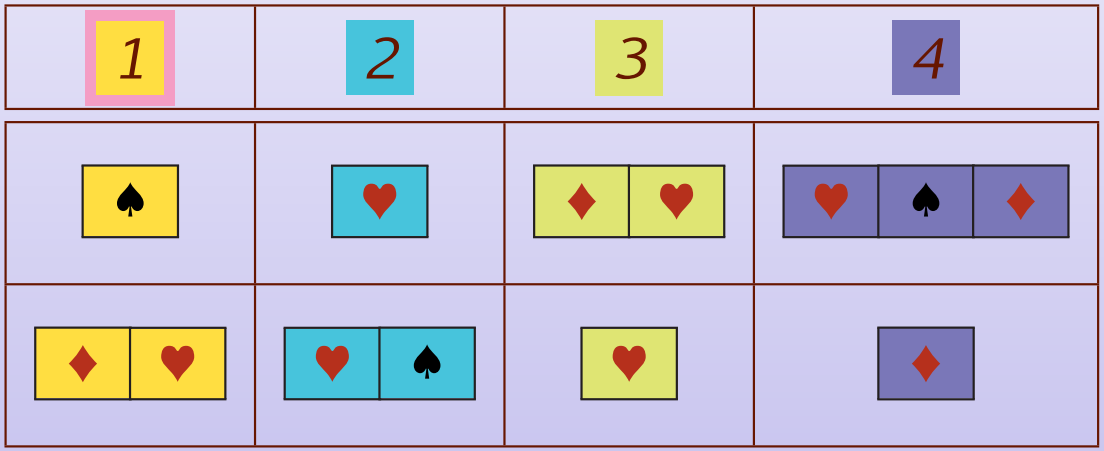
Other instance of the same kind of puzzle



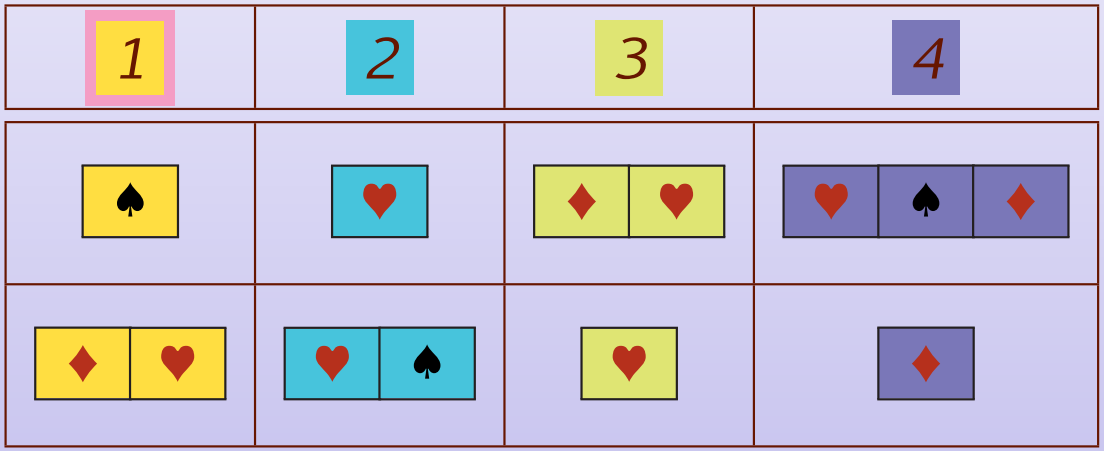
Other instance of the same kind of puzzle



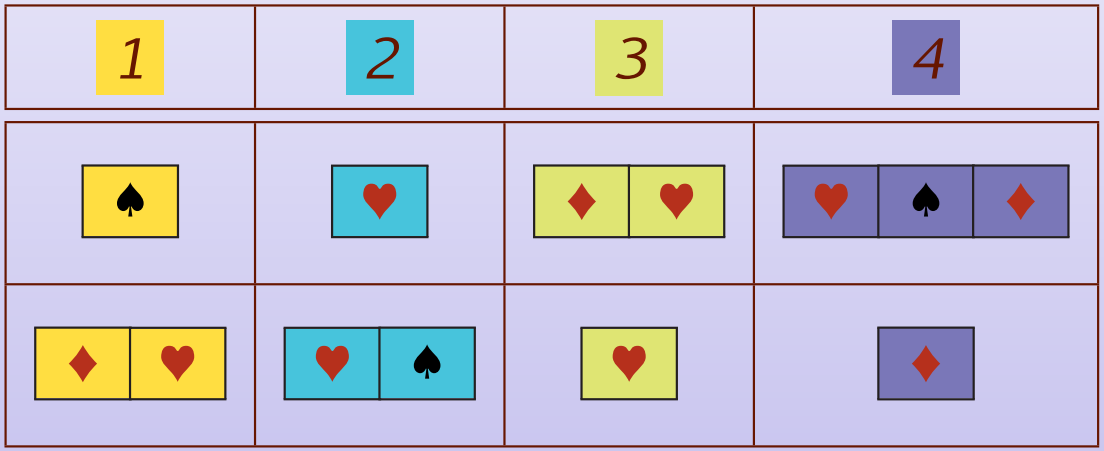
Other instance of the same kind of puzzle



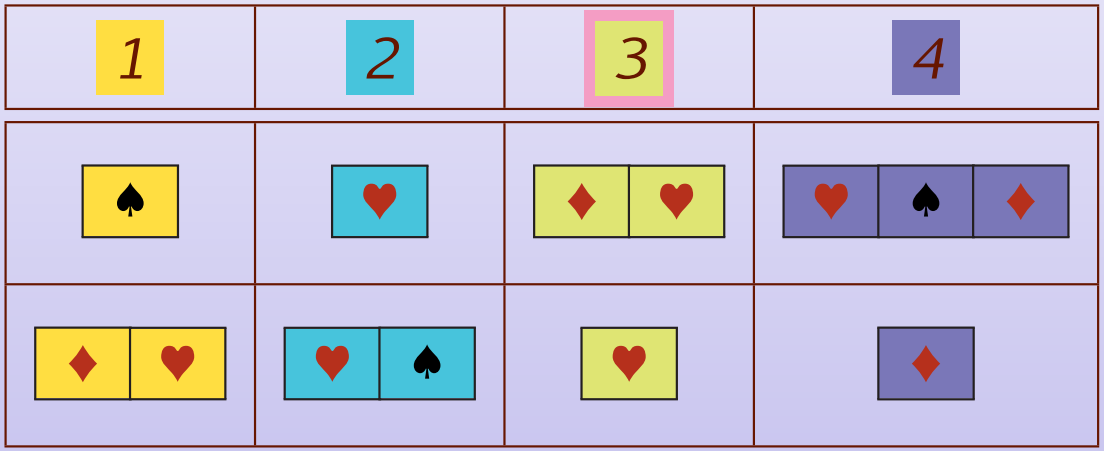
Other instance of the same kind of puzzle



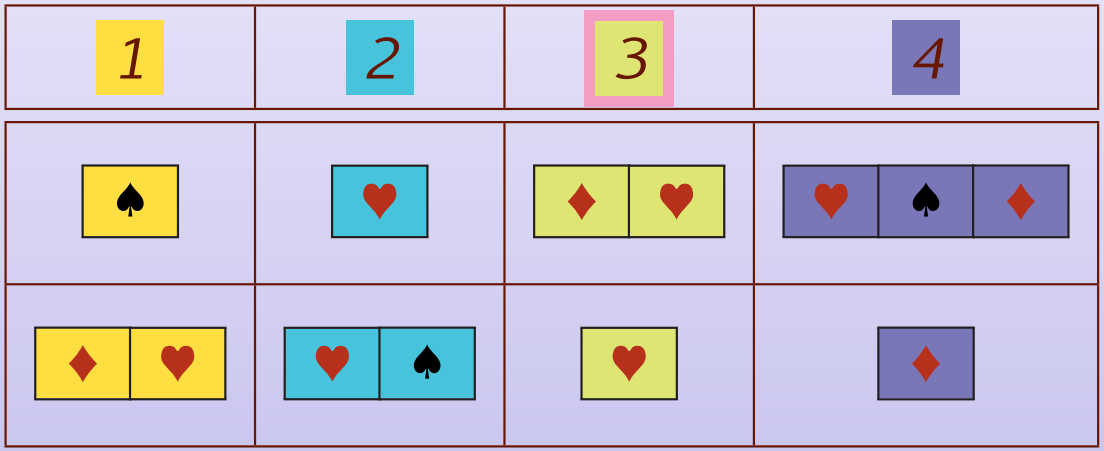
Other instance of the same kind of puzzle



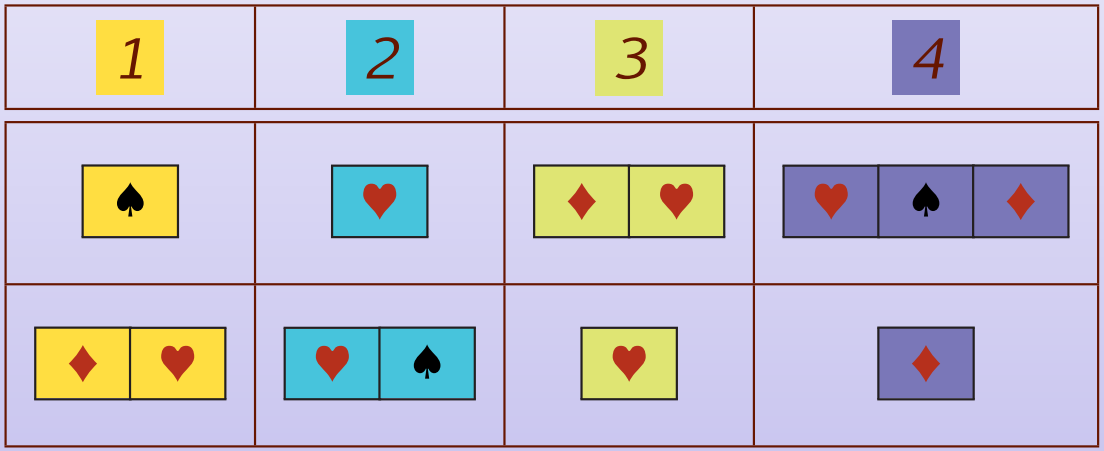
Other instance of the same kind of puzzle



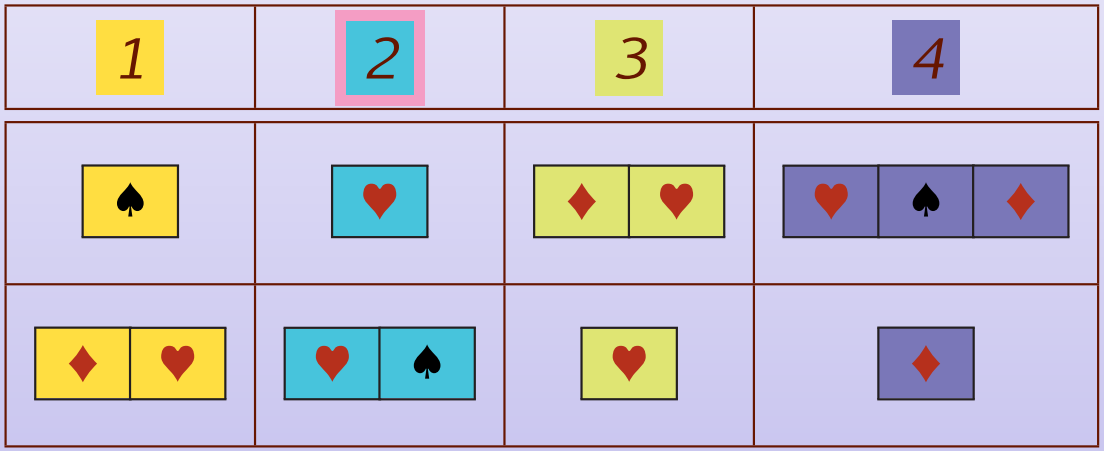
Other instance of the same kind of puzzle



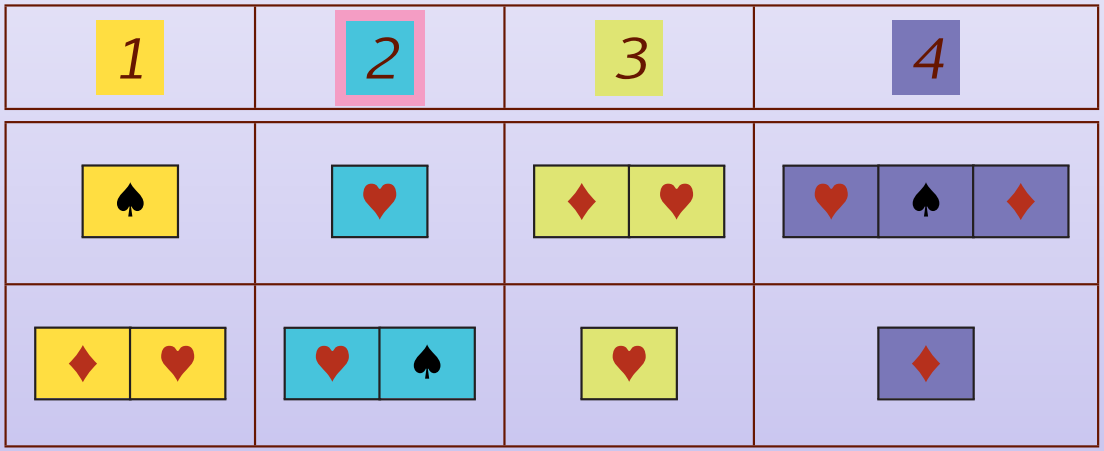
Other instance of the same kind of puzzle



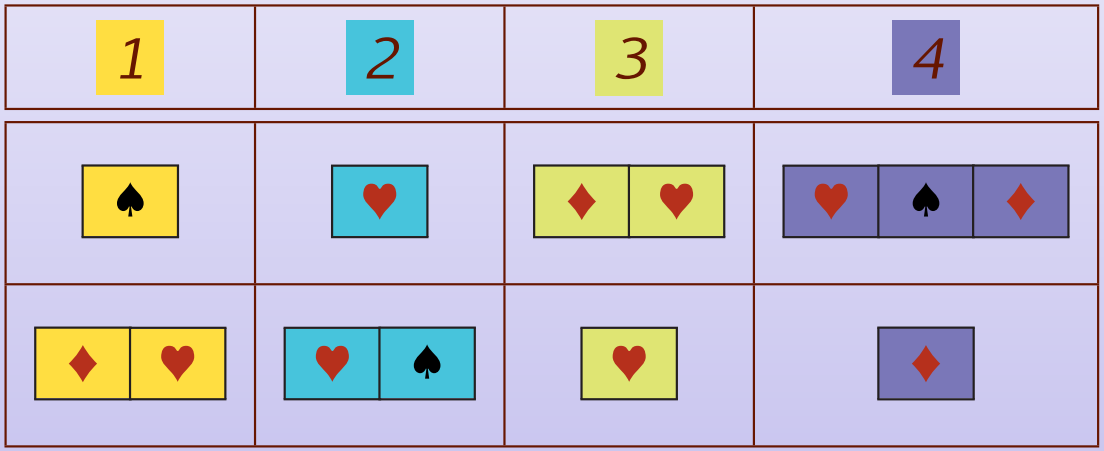
Other instance of the same kind of puzzle



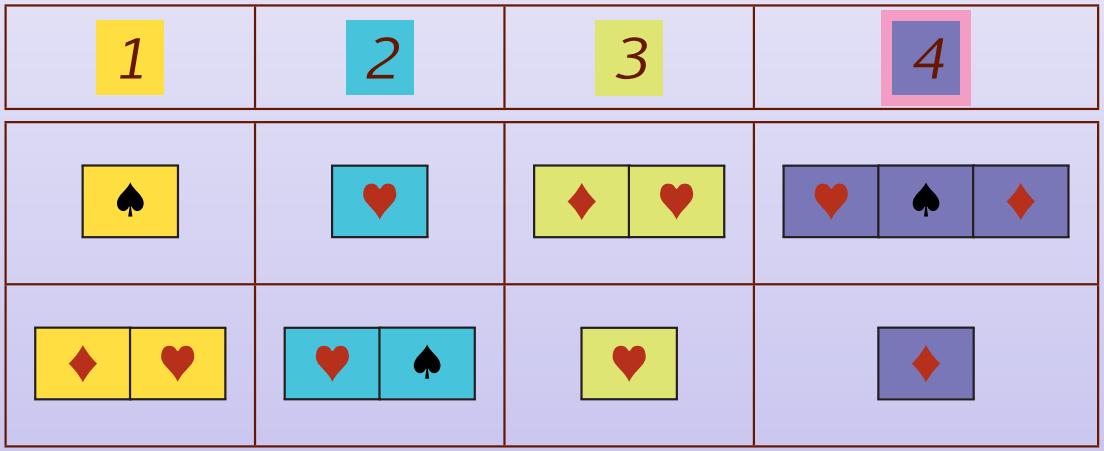
Other instance of the same kind of puzzle



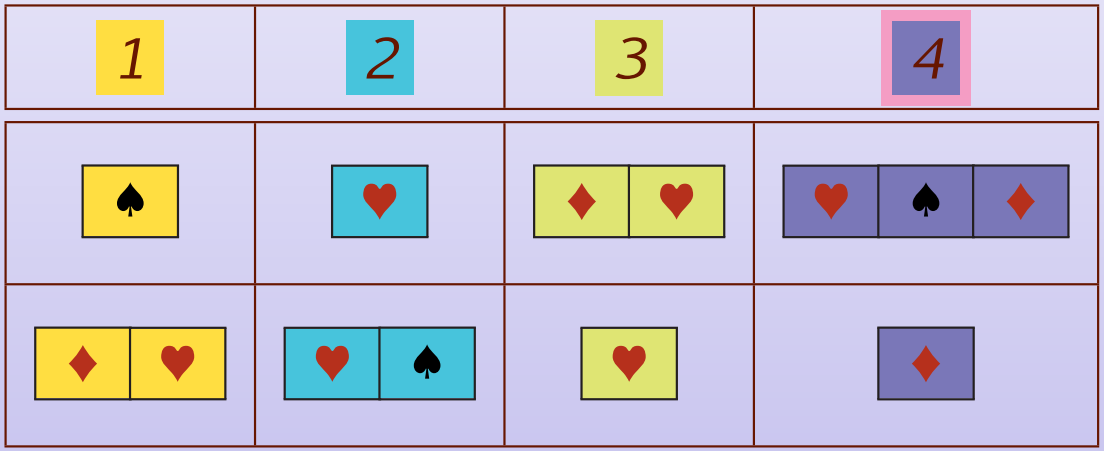
Other instance of the same kind of puzzle

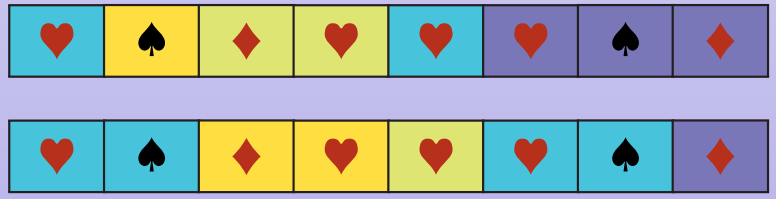


Other instance of the same kind of puzzle

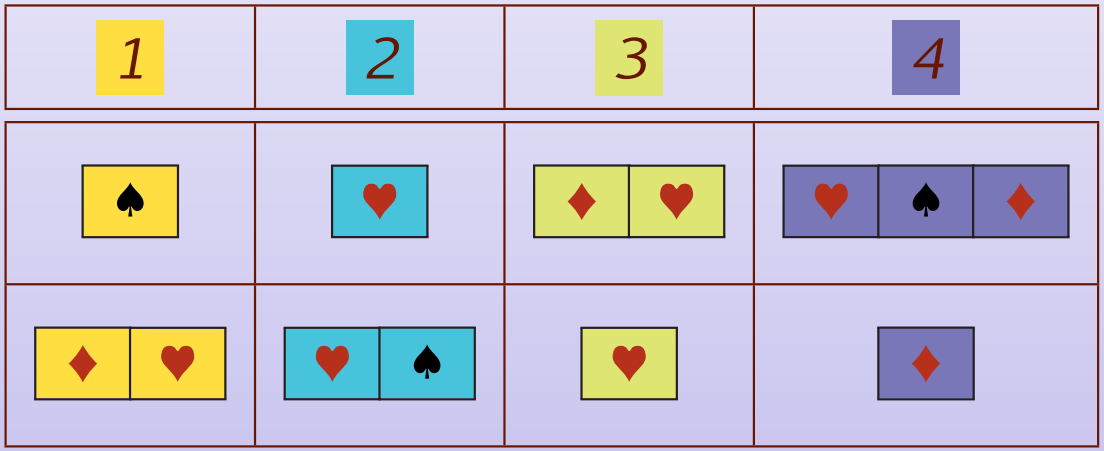


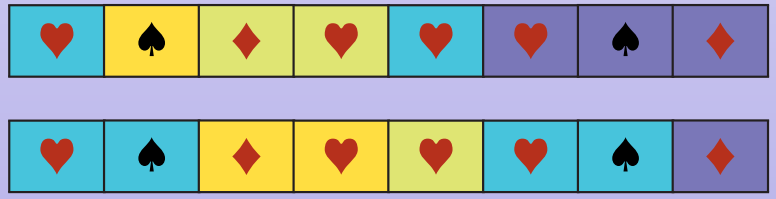
Other instance of the same kind of puzzle





Other instance of the same kind of puzzle

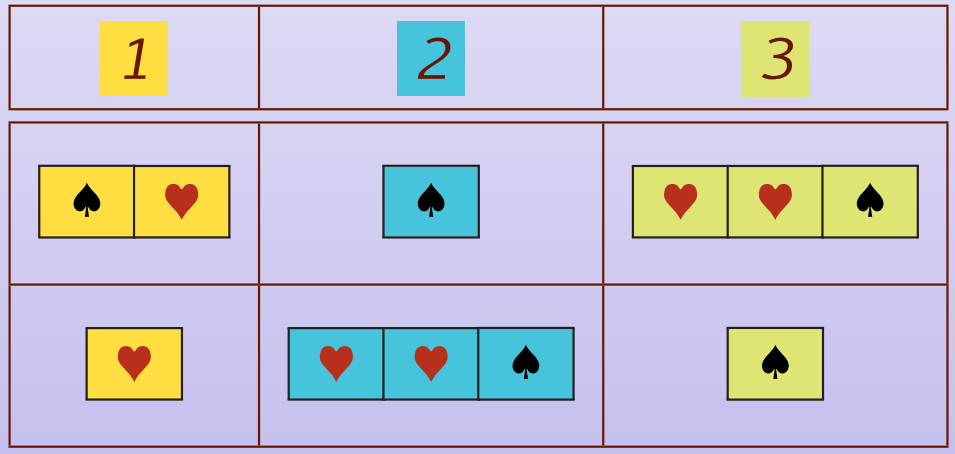




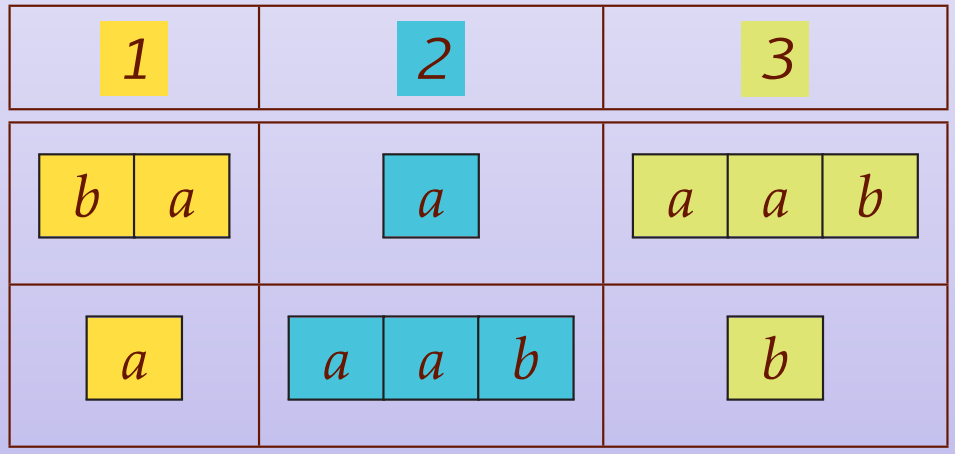
Done

Let's try another one (last play)

Let's try another one (last play)



Last play



Last play

1	2	3
ba	a	aab
a	aab	b

Last play

1	2	3
ba	a	aab
a	aab	b

We have a list of 3 pairs of words : $(ba,a),(a,aab),(aab,b)$.

In general, an instance is given be $n \in \mathbb{N}$ pairs of words.

Write a program

Exercise

Write a program which takes as input a list of pairs of words and which prints

Write a program

Exercise

Write a program which takes as input a list of pairs of words and which prints

- $\boxed{\textit{yes}}$ *if there exists a solution*

Write a program

Exercise

Write a program which takes as input a list of pairs of words and which prints

- $\boxed{\textit{yes}}$ *if there exists a solution*
- $\boxed{\textit{no}}$ *if there is no solution*

Post's correspondence problem

Emil Post

1897 (Poland) – 1954 (USA)

Post's correspondence problem

Post's correspondence problem

In 1941 Post established a theorem

Post's correspondence problem

In 1941 Post established a theorem

which states that no algorithm can exist which takes as input any list of pairs of words and outputs, after a finite time

Post's correspondence problem

In 1941 Post established a theorem

which states that no algorithm can exist which takes as input any list of pairs of words and outputs, after a finite time

- yes *if the corresponding puzzle has a solution*

Post's correspondence problem

In 1941 Post established a theorem

which states that no algorithm can exist which takes as input any list of pairs of words and outputs, after a finite time

- yes *if the corresponding puzzle has a solution*
- no *if there is no such solution.*

Post's correspondence problem

In 1941 Post established a theorem

which states that no algorithm can exist which takes as input any list of pairs of words and outputs, after a finite time

- yes *if the corresponding puzzle has a solution*
- no *if there is no such solution.*

Post's correspondence problem

In 1941 Post established a theorem

which states that no algorithm can exist which takes as input any list of pairs of words and outputs, after a finite time

- yes *if the corresponding puzzle has a solution*
- no *if there is no such solution.*

Problems for which no algorithm can exist are called

u n d e c i d a b l e

Decision problems

Decision problems

Adjectives

decidable

and

undecidable

Decision problems

Adjectives decidable
and undecidable

concern problems with a question which should be answered by «yes» or «no».

Decision problems

Adjectives decidable and undecidable concern problems with a question which should be answered by «yes» or «no».

These are called

decision problems.

Decision problems

Adjectives decidable
and undecidable

concern problems with a question which should be answered by «yes» or «no».

These are called

decision problems.

Example

Input : *a natural number* $n \in \mathbb{N}$
Question : *is* n *prime ?*

Evaluation or construction problems

Evaluation or construction problems

consist in producing an object which fulfils problem's statement.

Evaluation or construction problems

consist in producing an object which fulfils problem's statement.

Example

Input : *a natural number* $n \in \mathbb{N}$

Output : *a decomposition of* n *into its prime factors*

Evaluation or construction problems

consist in producing an object which fulfils problem's statement.

Example

Input : *a natural number* $n \in \mathbb{N}$
Output : *a decomposition of* n *into its prime factors*

Such problems are either computable
or uncomputable

Evaluation or construction problems

consist in producing an object which fulfils problem's statement.

Example

Input : *a natural number* $n \in \mathbb{N}$
Output : *a decomposition of* n *into its prime factors*

Such problems are either computable or uncomputable

Also, any function is either computable or uncomputable

Decidability and computability

Decidability and computability

One may think that

Decidability and computability

One may think that

1. whether a problem is decidable or undecidable

Decidability and computability

One may think that

1. whether a problem is decidable or undecidable
2. whether a function is computable or uncomputable

Decidability and computability

One may think that

1. whether a problem is decidable or undecidable
2. whether a function is computable or uncomputable

depends on the computing device we consider

Models of computation

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine
- λ-calculus

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine
- λ-calculus
- (semi-) Thue systems

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine
- λ-calculus
- (semi-) Thue systems
- Post canonical systems

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine
- λ-calculus
- (semi-) Thue systems
- Post canonical systems
- Von Neumann machine (computer)

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine
- λ-calculus
- (semi-) Thue systems
- Post canonical systems
- Von Neumann machine (computer)
- Markov algorithms

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine
- λ-calculus
- (semi-) Thue systems
- Post canonical systems
- Von Neumann machine (computer)
- Markov algorithms
- Kolmogorov-Uspensky machine

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine
- λ-calculus
- (semi-) Thue systems
- Post canonical systems
- Von Neumann machine (computer)
- Markov algorithms
- Kolmogorov-Uspensky machine
- Minsky machine

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine
- λ-calculus
- (semi-) Thue systems
- Post canonical systems
- Von Neumann machine (computer)
- Markov algorithms
- Kolmogorov-Uspensky machine
- Minsky machine
- neural networks

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine
- λ-calculus
- (semi-) Thue systems
- Post canonical systems
- Von Neumann machine (computer)
- Markov algorithms
- Kolmogorov-Uspensky machine
- Minsky machine
- neural networks
- quantum models

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine
- λ-calculus
- (semi-) Thue systems
- Post canonical systems
- Von Neumann machine (computer)
- Markov algorithms
- Kolmogorov-Uspensky machine
- Minsky machine
- neural networks
- quantum models
- molecular models

Models of computation

Any physical device such as a (usual) computer corresponds to some

model of computation

- Turing machine
- Post machine
- λ-calculus
- (semi-) Thue systems
- Post canonical systems
- Von Neumann machine (computer)
- Markov algorithms
- Kolmogorov-Uspensky machine
- Minsky machine
- neural networks
- quantum models
- molecular models

Church-Turing Thesis

Alonzo Church

1903 – 1995

Church-Turing Thesis

Alonzo Church

1903 – 1995

Church-Turing Thesis (1936)

Church-Turing Thesis

Alonzo Church

1903 – 1995

Church-Turing Thesis (1936)

All feasible (unrestricted) models of computation lead to the same notion of computability.

Church-Turing Thesis

Alonzo Church

1903 – 1995

Church-Turing Thesis (1936)

All feasible (unrestricted) models of computation lead to the same notion of computability.

No physical device can solve an undecidable problem or compute a function which is not computable.

Entscheidungsproblem (1928)

David Hilbert

1862 – 1943

Wilhelm Ackermann

1896 – 1962

Entscheidungsproblem (1928)

David Hilbert

1862 – 1943

Wilhelm Ackermann

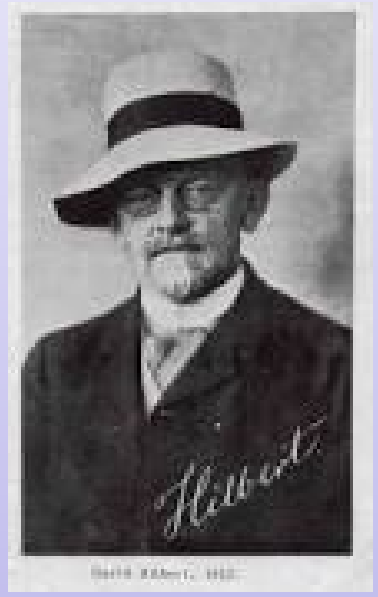
1896 – 1962

stated in 1928 «The Decision Problem» or «*Entscheidungsproblem*».

Entscheidungsproblem (1928)

David Hilbert

1862 – 1943



Wilhelm Ackermann

1896 – 1962

stated in 1928 «The Decision Problem» or «*Entscheidungsproblem*».

Input: a set of 1st order axioms Φ and a 1st order formula ψ.

Question: $\Phi \vDash \psi$?

The basic undecidable problem

Alan Turing

1912 – 1954

The basic undecidable problem

Alan Turing

1912 – 1954

1936 the concept of undecidability and a negative answer to *Entscheidungsproblem*

The basic undecidable problem

Alan Turing

 1912 – 1954

1936 the concept of undecidability and a negative answer to *Entscheidungsproblem*

1939-1943 deciphering German military messages encrypted with enhanced Enigma machine

The basic undecidable problem

Alan Turing

1912 – 1954

1936 the concept of undecidability and a negative answer to *Entscheidungsproblem*

1939-1943 deciphering German military messages encrypted with enhanced Enigma machine

1950 the concept of artificial intelligence

The basic undecidable problem

Alan Turing

1912 – 1954

1936 the concept of undecidability and a negative answer to *Entscheidungsproblem*

1939-1943 deciphering German military messages encrypted with enhanced Enigma machine

1950 the concept of artificial intelligence

1952 work on morphogenesis: reaction-diffusion systems, Turing patterns

The basic undecidable problem

The basic undecidable problem

One symbol is reserved as separation, for instance «£».

The basic undecidable problem

One symbol is reserved as separation, for instance «£».

We only consider programs which take as input a string (viz., a finite sequence of characters) and, upon termination, return

yes or no

The basic undecidable problem

One symbol is reserved as separation, for instance «£».

We only consider programs which take as input a string (viz., a finite sequence of characters) and, upon termination, return

yes or no

However, a program may also not terminate. It then returns nothing.

The basic undecidable problem

One symbol is reserved as separation, for instance «£».

We only consider programs which take as input a string (viz., a finite sequence of characters) and, upon termination, return

yes or no

However, a program may also not terminate. It then returns nothing.

Notation Let $\mathscr{P}$ be the set of all such programs.

The basic undecidable problem

Observations

The basic undecidable problem

Observations

- Each program is a string.

The basic undecidable problem

Observations

- Each program is a string.
- A program $P \in \mathscr{P}$ together with an input string w may be written as single string

$$P£w \ .$$

The basic undecidable problem

Observations

- Each program is a string.
- A program $P \in \mathscr{P}$ together with an input string w may be written as single string

$$P£w \ .$$

The basic undecidable problem

Observations

- Each program is a string.
- A program $P \in \mathscr{P}$ together with an input string w may be written as single string

$$P£w \ .$$

Notation

The basic undecidable problem

Observations

- Each program is a string.
- A program $P \in \mathscr{P}$ together with an input string w may be written as single string

$$P£w \ .$$

Notation

The result $\left(\boxed{\text{yes}} \text{ or } \boxed{\text{no}}\right)$ of a call of P on input w is written $P(w)$.

The basic undecidable problem

Observations

- Each program is a string.
- A program $P \in \mathscr{P}$ together with an input string w may be written as single string

$$P£w \ .$$

Notation

The result $\left(\boxed{\text{yes}} \text{ or } \boxed{\text{no}}\right)$ of a call of P on input w is written $P(w)$.

Remark

The basic undecidable problem

Observations

- Each program is a string.
- A program $P \in \mathscr{P}$ together with an input string w may be written as single string

$$P£w \ .$$

Notation

The result $\left(\boxed{\text{yes}} \text{ or } \boxed{\text{no}}\right)$ of a call of P on input w is written $P(w)$.

Remark

Either assertion

$$P(w) = \boxed{\text{yes}}$$

or

$$P(w) = \boxed{\text{no}}$$

implies that the running time of P on input w is finite.

The basic undecidable problem

The membership problem
(for recursively enumerable languages)

Input : a program $P \in \mathscr{P}$ and a string w
Question : $P(w) = \boxed{\text{yes}}$?

The basic undecidable problem

The membership problem
(for recursively enumerable languages)

Input : a program $P \in \mathscr{P}$ and a string w
Question : $P(w) = \boxed{\text{yes}}$?

Theorem (Turing 1936)
The membership problem undecidable.

Proof of the theorem

Assume by contradiction that the membership problem is decidable.

Proof of the theorem

Assume by contradiction that the membership problem is decidable. Then, there is an algorithm implemented as some program $R \in \mathscr{P}$ which takes as input

Proof of the theorem

Assume by contradiction that the membership problem is decidable. Then, there is an algorithm implemented as some program $R \in \mathscr{P}$ which takes as input

- any program $P \in \mathscr{P}$ and

Proof of the theorem

Assume by contradiction that the membership problem is decidable. Then, there is an algorithm implemented as some program $R \in \mathscr{P}$ which takes as input

- any program $P \in \mathscr{P}$ and
- any string w

Proof of the theorem

Assume by contradiction that the membership problem is decidable. Then, there is an algorithm implemented as some program $R \in \mathscr{P}$ which takes as input

- any program $P \in \mathscr{P}$ and
- any string w

under the form of a single string $P£w$,

Proof of the theorem

Assume by contradiction that the membership problem is decidable. Then, there is an algorithm implemented as some program $R \in \mathscr{P}$ which takes as input

- any program $P \in \mathscr{P}$ and
- any string w

under the form of a single string $P£w$,

which always terminates and returns

Proof of the theorem

Assume by contradiction that the membership problem is decidable. Then, there is an algorithm implemented as some program $R \in \mathscr{P}$ which takes as input

- any program $P \in \mathscr{P}$ and
- any string w

under the form of a single string $P£w$,

which always terminates and returns

- $\boxed{\text{yes}}$ if $P(w) = \boxed{\text{yes}}$

Proof of the theorem

Assume by contradiction that the membership problem is decidable. Then, there is an algorithm implemented as some program $R \in \mathscr{P}$ which takes as input

- any program $P \in \mathscr{P}$ and
- any string w

under the form of a single string $P£w$,

which always terminates and returns

- $\boxed{\text{yes}}$ if $P(w) = \boxed{\text{yes}}$
- $\boxed{\text{no}}$ otherwise.

Proof of the theorem

Assume by contradiction that the membership problem is decidable. Then, there is an algorithm implemented as some program $R \in \mathscr{P}$ which takes as input

- any program $P \in \mathscr{P}$ and
- any string w

under the form of a single string $P£w$,

which always terminates and returns

- $\boxed{\text{yes}}$ if $P(w) = \boxed{\text{yes}}$
- $\boxed{\text{no}}$ otherwise.

In case when P does not terminate on input w

Proof of the theorem

Assume by contradiction that the membership problem is decidable. Then, there is an algorithm implemented as some program $R \in \mathscr{P}$ which takes as input

- any program $P \in \mathscr{P}$ and
- any string w

under the form of a single string $P£w$,

which always terminates and returns

- $\boxed{\text{yes}}$ if $P(w) = \boxed{\text{yes}}$
- $\boxed{\text{no}}$ otherwise.

In case when P does not terminate on input w R returns $\boxed{\text{no}}$.

Proof of the theorem

Assume by contradiction that the membership problem is decidable. Then, there is an algorithm implemented as some program $R \in \mathscr{P}$ which takes as input

- any program $P \in \mathscr{P}$ and
- any string w

under the form of a single string $P£w$,

which always terminates and returns

- $\boxed{\text{yes}}$ if $P(w) = \boxed{\text{yes}}$
- $\boxed{\text{no}}$ otherwise.

In case when P does not terminate on input w R returns $\boxed{\text{no}}$.

If the input is not of the form $P£w$,

Proof of the theorem

Assume by contradiction that the membership problem is decidable. Then, there is an algorithm implemented as some program $R \in \mathscr{P}$ which takes as input

- any program $P \in \mathscr{P}$ and
- any string w

under the form of a single string $P£w$,

which always terminates and returns

- $\boxed{\text{yes}}$ if $P(w) = \boxed{\text{yes}}$
- $\boxed{\text{no}}$ otherwise.

In case when P does not terminate on input w
R returns $\boxed{\text{no}}$.

If the input is not of the form $P£w$,
R also returns $\boxed{\text{no}}$.

Proof of the theorem

We write the following program, which calls R:

Proof of the theorem

We write the following program, which calls R:

```
program S(P:string)
if   R(P£P) = yes
                    return no
else
                    return yes
```

Proof of the theorem

We write the following program, which calls R:

```
program S(P:string)
if   R(P£P) = yes
                    return no
else
                    return yes
```

We wish to know what S returns when called with input S.

Proof of the theorem

We write the following program, which calls R:

```
program S(P:string)
if   R(P£P) = yes
                    return no
else
                    return yes
```

We wish to know what S returns when called with input S.

case 1 $S(S) =$ yes.

Proof of the theorem

We write the following program, which calls R:

```
program S(P : string)
if   R(P£P) = yes
                    return no
else
                    return yes
```

We wish to know what S returns when called with input S.

case 1 $S(S) =$ yes. Then $R(S£S) =$ no.

Proof of the theorem

We write the following program, which calls R:

```
program S(P : string)
if   R(P£P) = yes
                    return no
else
                    return yes
```

We wish to know what S returns when called with input S.

case 1 $S(S) = \boxed{\text{yes}}$. Then $R(S£S) = \boxed{\text{no}}$. Hence $S(S) = \boxed{\text{no}}$.

Proof of the theorem

We write the following program, which calls R:

```
program S(P:string)
if   R(P£P) = yes
                    return no
else
                    return yes
```

We wish to know what S returns when called with input S.

case 1 $S(S) = \boxed{\text{yes}}$. Then $R(S£S) = \boxed{\text{no}}$. Hence $S(S) = \boxed{\text{no}}$. Contradiction.

Proof of the theorem

We write the following program, which calls R:

```
program S(P : string)
if   R(P£P) = yes
                    return no
else
                    return yes
```

We wish to know what S returns when called with input S.

case 1 $S(S) = \boxed{\text{yes}}$. Then $R(S£S) = \boxed{\text{no}}$. Hence $S(S) = \boxed{\text{no}}$. Contradiction.

case 2 $S(S) = \boxed{\text{no}}$.

Proof of the theorem

We write the following program, which calls R:

$$\begin{array}{l}\texttt{program}\ S(P:\texttt{string})\\ \texttt{if}\quad R(P\pounds P) = \boxed{\text{yes}}\\ \qquad\qquad\qquad\qquad \texttt{return}\ \boxed{\text{no}}\\ \texttt{else}\\ \qquad\qquad\qquad\qquad \texttt{return}\ \boxed{\text{yes}}\end{array}$$

We wish to know what S returns when called with input S.

case 1 $S(S) = \boxed{\text{yes}}$. Then $R(S\pounds S) = \boxed{\text{no}}$. Hence $S(S) = \boxed{\text{no}}$. Contradiction.

case 2 $S(S) = \boxed{\text{no}}$. Then $R(S\pounds S) = \boxed{\text{yes}}$.

Proof of the theorem

We write the following program, which calls R:

```
program S(P:string)
if   R(P£P) = yes
                    return no
else
                    return yes
```

We wish to know what S returns when called with input S.

case 1 $S(S) = \boxed{\text{yes}}$. Then $R(S£S) = \boxed{\text{no}}$. Hence $S(S) = \boxed{\text{no}}$. Contradiction.

case 2 $S(S) = \boxed{\text{no}}$. Then $R(S£S) = \boxed{\text{yes}}$. Hence $S(S) = \boxed{\text{yes}}$.

Proof of the theorem

We write the following program, which calls R:

```
program S(P:string)
if   R(P£P) = yes
                    return no
else
                    return yes
```

We wish to know what S returns when called with input S.

case 1 $S(S) = \boxed{\text{yes}}$. Then $R(S£S) = \boxed{\text{no}}$. Hence $S(S) = \boxed{\text{no}}$. Contradiction.

case 2 $S(S) = \boxed{\text{no}}$. Then $R(S£S) = \boxed{\text{yes}}$. Hence $S(S) = \boxed{\text{yes}}$. Contradiction.

Proof of the theorem

We write the following program, which calls R:

```
program S(P:string)
if   R(P£P) = yes
                    return no
else
                    return yes
```

We wish to know what S returns when called with input S.

case 1 $S(S) = \boxed{\text{yes}}$. Then $R(S£S) = \boxed{\text{no}}$. Hence $S(S) = \boxed{\text{no}}$. Contradiction.

case 2 $S(S) = \boxed{\text{no}}$. Then $R(S£S) = \boxed{\text{yes}}$. Hence $S(S) = \boxed{\text{yes}}$. Contradiction.

Problem reduction

Instead of proving undecidability of a problem from scratch, it is often convenient to proceed through problem reduction

Problem reduction

Instead of proving undecidability of a problem from scratch, it is often convenient to proceed through problem reduction

Definition

$P_1 = \langle E_1, \text{Yes}_1 \rangle$ is many-one reducible to $P_2 = \langle E_2, \text{Yes}_2 \rangle$,

Problem reduction

Instead of proving undecidability of a problem from scratch, it is often convenient to proceed through problem reduction

Definition

$P_1 = \langle E_1, \text{Yes}_1 \rangle$ is many-one reducible to $P_2 = \langle E_2, \text{Yes}_2 \rangle$,

if there exists a computable map $\rho : E_1 \to E_2$

Problem reduction

Instead of proving undecidability of a problem from scratch, it is often convenient to proceed through problem reduction

Definition

$P_1 = \langle E_1, \text{Yes}_1 \rangle$ is many-one reducible to $P_2 = \langle E_2, \text{Yes}_2 \rangle$,

if there exists a computable map $\rho : E_1 \to E_2$

such that, for all $e \in E_1$

$$e \in \text{Yes}_1 \quad \Leftrightarrow \quad \rho(e) \in \text{Yes}_2$$

Problem reduction

Instead of proving undecidability of a problem from scratch, it is often convenient to proceed through problem reduction

Definition

$P_1 = \langle E_1, \text{Yes}_1 \rangle$ is many-one reducible to $P_2 = \langle E_2, \text{Yes}_2 \rangle$,

if there exists a computable map $\rho : E_1 \to E_2$

such that, for all $e \in E_1$

$$e \in \text{Yes}_1 \quad \Leftrightarrow \quad \rho(e) \in \text{Yes}_2$$

$$\begin{aligned} P_1 \text{ undecidable} &\Rightarrow P_2 \text{ undecidable} \\ P_2 \text{ decidable} &\Rightarrow P_1 \text{ decidable} \end{aligned}$$

Problem reduction (example)

We may reduce Post's correspondence problem into the membership problem.

Problem reduction (example)

We may reduce Post's correspondence problem into the membership problem.

- To every instance of Post's correspondence problem we associate a program which tries all sequences of pairs of words in canonical order.

Problem reduction (example)

We may reduce Post's correspondence problem into the membership problem.

- To every instance of Post's correspondence problem we associate a program which tries all sequences of pairs of words in canonical order.
- It is easy to write a program suitable for all instances.

Problem reduction (example)

We may reduce Post's correspondence problem into the membership problem.

- To every instance of Post's correspondence problem we associate a program which tries all sequences of pairs of words in canonical order.
- It is easy to write a program suitable for all instances.

Problem reduction (example)

We may reduce Post's correspondence problem into the membership problem.

- To every instance of Post's correspondence problem we associate a program which tries all sequences of pairs of words in canonical order.
- It is easy to write a program suitable for all instances.

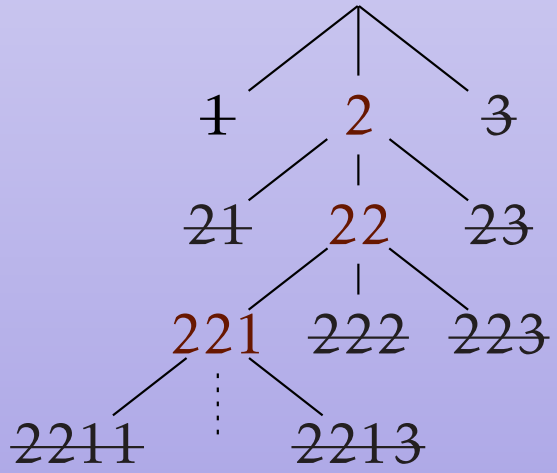
1	2	3
ba	a	aab
a	aab	b

Problem reduction (example)

We may reduce Post's correspondence problem into the membership problem.

- To every instance of Post's correspondence problem we associate a program which tries all sequences of pairs of words in canonical order.
- It is easy to write a program suitable for all instances.

1	2	3
ba	a	aab
a	aab	b



Hilbert's tenth problem

Input: a multivariate polynomial $p(x_1,\ldots,x_n)$ with integer coefficients.

Question: Has

$$p(x_1,\ldots,x_n) = 0 \qquad \text{(a Diophantine equation)}$$

a solution in $\mathbb{Z}^n$?

Hilbert's tenth problem

Input: a multivariate polynomial $p(x_1,\ldots,x_n)$ with integer coefficients.

Question: Has

$$p(x_1,\ldots,x_n) = 0 \qquad \text{(a Diophantine equation)}$$

a solution in $\mathbb{Z}^n$?

Theorem (Matiyasevich 1970)

Hilbert's tenth problem is undecidable.

Problems involving real numbers

Let Φ_1 be a set of formula with real variables built using

Problems involving real numbers

Let Φ_1 be a set of formula with real variables built using

- 1, "$+$", "$\cdot$"

Problems involving real numbers

Let Φ_1 be a set of formula with real variables built using

- 1, "$+$", "$\cdot$"
- "$\leq$"

Problems involving real numbers

Let Φ_1 be a set of formula with real variables built using

- 1, "+", "$\cdot$"
- "$\leq$"
- logical connectives ("$\Rightarrow$", "$\neg$", "$\wedge$", etc.),

Problems involving real numbers

Let Φ_1 be a set of formula with real variables built using

- 1, "+", "$\cdot$"
- "$\leq$"
- logical connectives ("$\Rightarrow$", "$\neg$", "$\wedge$", etc.),
- quantifiers ("$\exists$", "$\forall$").

Problems involving real numbers

Let Φ_1 be a set of formula with real variables built using

- 1, "+", "$\cdot$"
- "$\leq$"
- logical connectives ("$\Rightarrow$", "$\neg$", "$\wedge$", etc.),
- quantifiers ("$\exists$", "$\forall$").

Problems involving real numbers

Let Φ_1 be a set of formula with real variables built using

- 1, "+", "·"
- "$\leq$"
- logical connectives ("$\Rightarrow$", "$\neg$", "$\wedge$", etc.),
- quantifiers ("$\exists$", "$\forall$").

Theorem (Tarski 1930)

The following problem

Input : $\psi \in \Phi_1$

Question : *Does* ψ *hold in the field real numbers ?*

Problems involving real numbers

Let Φ_1 be a set of formula with real variables built using

- 1, "+", "·"
- "$\leq$"
- logical connectives ("$\Rightarrow$", "$\neg$", "$\wedge$", etc.),
- quantifiers ("$\exists$", "$\forall$").

Theorem (Tarski 1930)

The following problem

Input : $\psi \in \Phi_1$

Question : *Does* ψ *hold in the field real numbers ?*

is decidable.

Problems involving real numbers

Let Φ_1 be a set of formula with real variables built using

- 1, "+", "·"
- "≤"
- logical connectives ("⇒", "¬", "∧", etc.),
- quantifiers ("∃", "∀").

Theorem (Tarski 1930)

The following problem

Input : $\psi \in \Phi_1$

Question : *Does* ψ *hold in the field real numbers ?*

is decidable.

$$\exists x_1 \ldots \exists x_n \; p(x_1, \ldots, x_n) = 0$$

Problems involving real numbers

Let $\mathscr{F}_0$ be a class of function of several real variables that can be constructed by composition of

- $1, \text{“}+\text{”}, \text{“}\cdot\text{”}, \text{“}\sin\text{”}$

Problems involving real numbers

Let $\mathscr{F}_0$ be a class of function of several real variables that can be constructed by composition of

- 1, "$+$", "$\cdot$", "$\sin$"

Theorem (Richardson, Caviness, Wang, Laczkovich)

The following problem

Input : $f(x_1, \ldots, x_n) \in \mathscr{F}_0$

Question : *Has*

$$f(x_1, \ldots, x_n) = 0$$

a real solution ?

Problems involving real numbers

Let $\mathscr{F}_0$ be a class of function of several real variables that can be constructed by composition of

- 1, "$+$", "$\cdot$", "$\sin$"

Theorem (Richardson, Caviness, Wang, Laczkovich)

The following problem

Input : $f(x_1, \ldots, x_n) \in \mathscr{F}_0$

Question : *Has*

$$f(x_1, \ldots, x_n) = 0$$

a real solution ?

is undecidable.

Problems involving real numbers

Let $\mathscr{F}_0$ be a class of function of several real variables that can be constructed by composition of

- 1, "$+$", "$\cdot$", "$\sin$"

Theorem (Richardson, Caviness, Wang, Laczkovich)

The following problem

Input : $f(x_1, \ldots, x_n) \in \mathscr{F}_0$

Question : *Has*

$$f(x_1, \ldots, x_n) = 0$$

a real solution ?

is undecidable.

Problems involving real numbers

Let $\mathscr{G}$ be a class of function of several real variables that can be constructed by composition of

- $1, \text{“}+\text{”}, \text{“}\cdot\text{”}, \text{“}\exp\text{”}$

Problems involving real numbers

Let $\mathscr{G}$ be a class of function of several real variables that can be constructed by composition of

- 1, "$+$", "$\cdot$", "exp"

Open problem

Input : $g(x_1,\ldots,x_n) \in \mathscr{G}$

Question : Has

$$g(x_1,\ldots,x_n) = 0$$

a real solution ?

Problems involving real numbers

Let $\mathscr{G}$ be a class of function of several real variables that can be constructed by composition of

- 1, "$+$", "$\cdot$", "exp"

Open problem

Input : $g(x_1,\ldots,x_n) \in \mathscr{G}$

Question : Has

$$g(x_1,\ldots,x_n) = 0$$

a real solution ?

Some consequences of Matiyasevich's Theorem

Theorem

The following problem

Input : *a system of ordinary differential equations*

$$\begin{cases} p_1(x, f_1(x), f_2(x), \ldots, f_n(x), f_1'(x)) = 0 \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \quad \vdots \\ p_n(x, f_1(x), f_2(x), \ldots, f_n(x), f_n'(x)) = 0 \end{cases}$$

Question : *Has the system a solution on the interval* $[0,1]$ *?*

Some consequences of Matiyasevich's Theorem

Theorem

The following problem

Input : *a system of ordinary differential equations*

$$\begin{cases} p_1(x, f_1(x), f_2(x), \ldots, f_n(x), f_1'(x)) = 0 \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \quad \vdots \\ p_n(x, f_1(x), f_2(x), \ldots, f_n(x), f_n'(x)) = 0 \end{cases}$$

Question : *Has the system a solution on the interval* $[0,1]$ *?*

is undecidable.

Some consequences of Matiyasevich's Theorem

Let $\mathscr{F}_3$ be a class of functions of one real variable that can be constructed by composition of

- $1, \text{“}+\text{”}, \text{“}-\text{”}, \text{“}\cdot\text{”}, \text{“}\div\text{”}, \text{“}\sin\text{”}$

Some consequences of Matiyasevich's Theorem

Let $\mathscr{F}_3$ be a class of functions of one real variable that can be constructed by composition of

- $1, "+", "-", "\cdot", "\div", "\sin"$

Theorem

The following problem

Input : a function $f \in \mathscr{F}_3$

Question : Does $\int_{-\infty}^{+\infty} f(x)\,dx$ converge ?

Some consequences of Matiyasevich's Theorem

Let $\mathscr{F}_3$ be a class of functions of one real variable that can be constructed by composition of

- $1, \text{“}+\text{”}, \text{“}-\text{”}, \text{“}\cdot\text{”}, \text{“}\div\text{”}, \text{“}\sin\text{”}$

Theorem

The following problem

Input : a function $f \in \mathscr{F}_3$

Question : Does $\int_{-\infty}^{+\infty} f(x)\,dx$ converge ?

is undecidable.

Some consequences of Matiyasevich's Theorem

Let $\mathscr{F}_2$ be a class of functions of one real variable that can be constructed by composition of

- 1, "$+$", "$-$", "$\cdot$", "$\div$", "$\sin$", absolute value

Some consequences of Matiyasevich's Theorem

Let $\mathscr{F}_2$ be a class of functions of one real variable that can be constructed by composition of

- 1, "$+$", "$-$", "$\cdot$", "$\div$", "$\sin$", absolute value

Let $\mathscr{F}_4$ and $\mathscr{F}_5$ be classes of functions of one real variable s.t.

- $\mathscr{F}_2 \subseteq \mathscr{F}_4$
- $\mathscr{F}_4$ is closed under multiplication
- At least one function in $\mathscr{F}_4$ has no antiderivative in $\mathscr{F}_5$.

Some consequences of Matiyasevich's Theorem

Let $\mathscr{F}_2$ be a class of functions of one real variable that can be constructed by composition of

- 1, "+", "−", "·", "÷", "sin", absolute value

Let $\mathscr{F}_4$ and $\mathscr{F}_5$ be classes of functions of one real variable s.t.

- $\mathscr{F}_2 \subseteq \mathscr{F}_4$
- $\mathscr{F}_4$ is closed under multiplication
- At least one function in $\mathscr{F}_4$ has no antiderivative in $\mathscr{F}_5$.

Theorem

The following problem

Input : a function $f \in \mathscr{F}_4$

Question : Is there $F \in \mathscr{F}_5$ s.t. $F'(x) = f(x)$?

Some consequences of Matiyasevich's Theorem

Let $\mathscr{F}_2$ be a class of functions of one real variable that can be constructed by composition of

- 1, "$+$", "$-$", "$\cdot$", "$\div$", "$\sin$", absolute value

Let $\mathscr{F}_4$ and $\mathscr{F}_5$ be classes of functions of one real variable s.t.

- $\mathscr{F}_2 \subseteq \mathscr{F}_4$
- $\mathscr{F}_4$ is closed under multiplication
- At least one function in $\mathscr{F}_4$ has no antiderivative in $\mathscr{F}_5$.

Theorem

The following problem

Input : a function $f \in \mathscr{F}_4$

Question : Is there $F \in \mathscr{F}_5$ s.t. $F'(x) = f(x)$?

is undecidable.

Some consequences of Matiyasevich's Theorem

- There are many methods for solving Diophantine equations of specific forms.

Some consequences of Matiyasevich's Theorem

- There are many methods for solving Diophantine equations of specific forms.
- However, some individual Diophantine equations may be very difficult to solve.

Some consequences of Matiyasevich's Theorem

- There are many methods for solving Diophantine equations of specific forms.
- However, some individual Diophantine equations may be very difficult to solve.

Some consequences of Matiyasevich's Theorem

- There are many methods for solving Diophantine equations of specific forms.
- However, some individual Diophantine equations may be very difficult to solve.

For every following conjecture/theorem, individual Diophantine equation has been constructed such that

Some consequences of Matiyasevich's Theorem

- There are many methods for solving Diophantine equations of specific forms.
- However, some individual Diophantine equations may be very difficult to solve.

For every following conjecture/theorem, individual Diophantine equation has been constructed such that

the conjecture/theorem holds	iff	the corresponding Diophantine equation has a solution

Some consequences of Matiyasevich's Theorem

- There are many methods for solving Diophantine equations of specific forms.
- However, some individual Diophantine equations may be very difficult to solve.

For every following conjecture/theorem, individual Diophantine equation has been constructed such that

the conjecture/theorem holds	iff	the corresponding Diophantine equation has a solution

- Goldbach's Conjecture,

Some consequences of Matiyasevich's Theorem

- There are many methods for solving Diophantine equations of specific forms.
- However, some individual Diophantine equations may be very difficult to solve.

For every following conjecture/theorem, individual Diophantine equation has been constructed such that

the conjecture/theorem holds	iff	the corresponding Diophantine equation has a solution

- Goldbach's Conjecture,
- Riemann's Hypothesis,

Some consequences of Matiyasevich's Theorem

- There are many methods for solving Diophantine equations of specific forms.
- However, some individual Diophantine equations may be very difficult to solve.

For every following conjecture/theorem, individual Diophantine equation has been constructed such that

the conjecture/theorem holds	iff	the corresponding Diophantine equation has a solution

- Goldbach's Conjecture,
- Riemann's Hypothesis,
- Fermat's Last Theorem,

Some consequences of Matiyasevich's Theorem

- There are many methods for solving Diophantine equations of specific forms.
- However, some individual Diophantine equations may be very difficult to solve.

For every following conjecture/theorem, individual Diophantine equation has been constructed such that

the conjecture/theorem holds	iff	the corresponding Diophantine equation has a solution

- Goldbach's Conjecture,
- Riemann's Hypothesis,
- Fermat's Last Theorem,
- Four Colour Theorem.

Problems that can be solved only for small inputs

In addition to undecidable problems, there are decidable problems which are untractable because they require an exponential amount of resources with respect to the input size.

Problems that can be solved only for small inputs

In addition to undecidable problems, there are decidable problems which are untractable because they require an exponential amount of resources with respect to the input size.

Example

Input: a finite set E, a set of mappings $\mathscr{F} \subseteq E^E$ and a map $f : E \to E$.

Question: Is f a composition of some mappings from $\mathscr{F}$?


Thank you
神威
太湖之光