Algorithmics

Lecture 10

Dr. Oana Andrei

School of Computing Science
University of Glasgow

oana.andrei@glasgow.ac.uk

Section 5 – Computability

Introduction

- unsolvable and undecidable problems
- the tiling problem
- Post's correspondence problem
- the halting problem

Models of computation

- finite-state automata
- pushdown automata
- Turing machines
- Counter machines
- Church-Turing thesis

Section 5 – Computability

Introduction

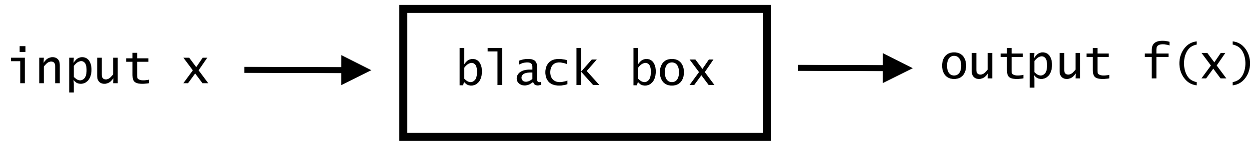
- unsolvable and undecidable problems
- the tiling problem
- Post's correspondence problem
- the halting problem

Models of computation

- finite-state automata – first part
- pushdown automata
- Turing machines
- Counter machines
- Church–Turing thesis

Introduction to Computability

What is a computer?



What can the black box do?

– it computes a function that maps an input to an output

Computability is concerned with which functions can be computed

– a formal way of answering ‘what problems can be solved by a computer?’
– or alternatively ‘what problems cannot be solved by a computer?’

To answer such questions we require a formal definition

– i.e. a definition of what a computer is
– or what an algorithm is if we view a computer as a device that can execute an algorithm

Unsolvable problems

Some problems cannot be solved by a computer

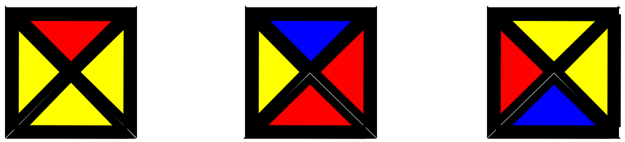
– even with unbounded time

Example: The Tiling Problem (decision problem)

– a tile is a 1×1 square, divided into 4 triangles by its diagonals with each triangle is given a colour

– each tile has a fixed orientation (no rotations allowed)

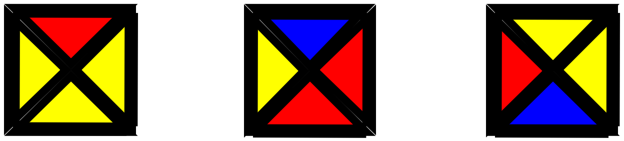
– example tiles:



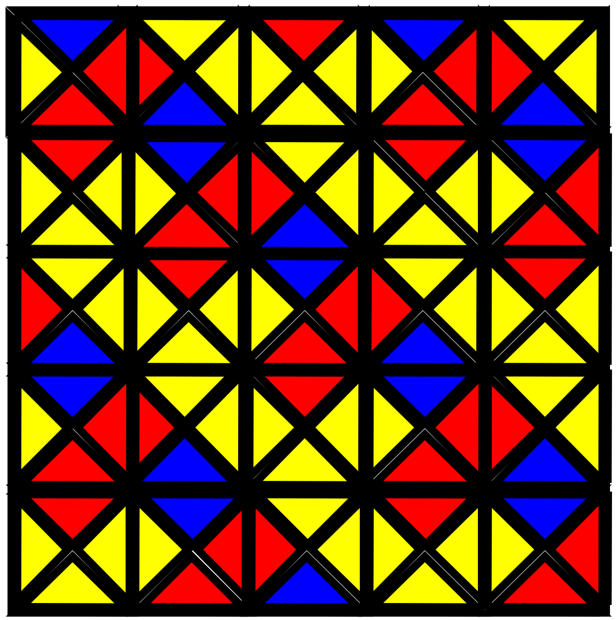
Instance: a finite set S of tile descriptions

Question: can any finite area, of any size, be completely covered using only tiles of types in S, so that adjacent tiles colour match?

Tiling problem – Tiling a 5 × 5 square

Available tiles: 

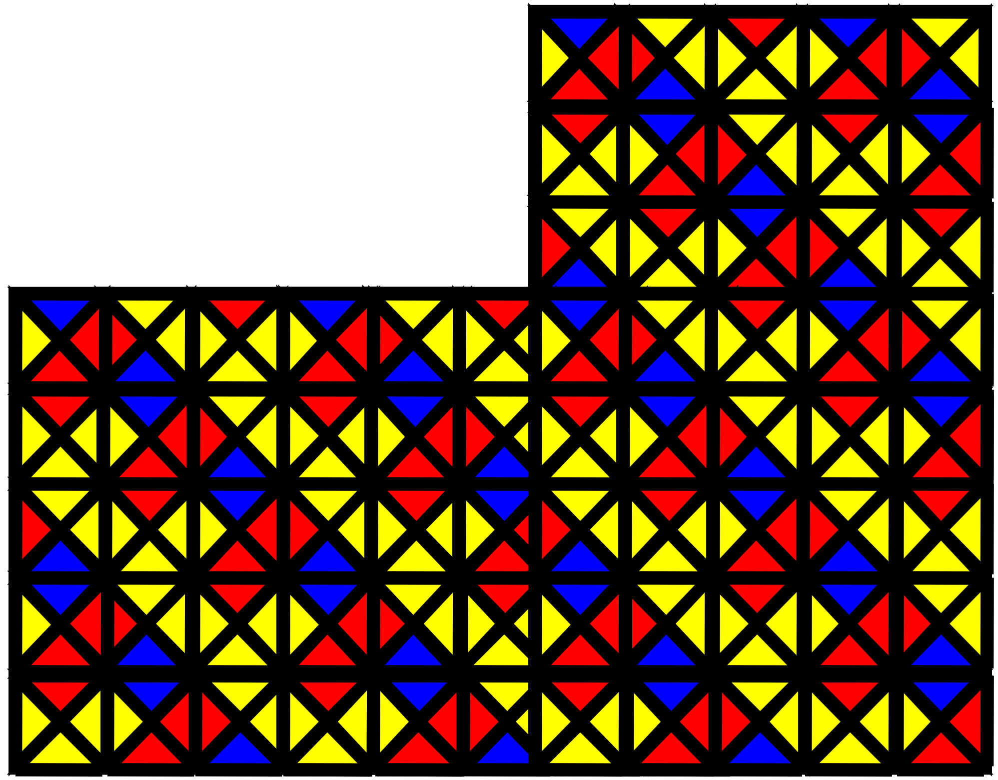
We can use these tiles to tile a 5 × 5 square as follows:



Tiling problem – Extending to a larger region

Overlap the top two rows with the bottom two rows

- obtain an 8×5 tiled area



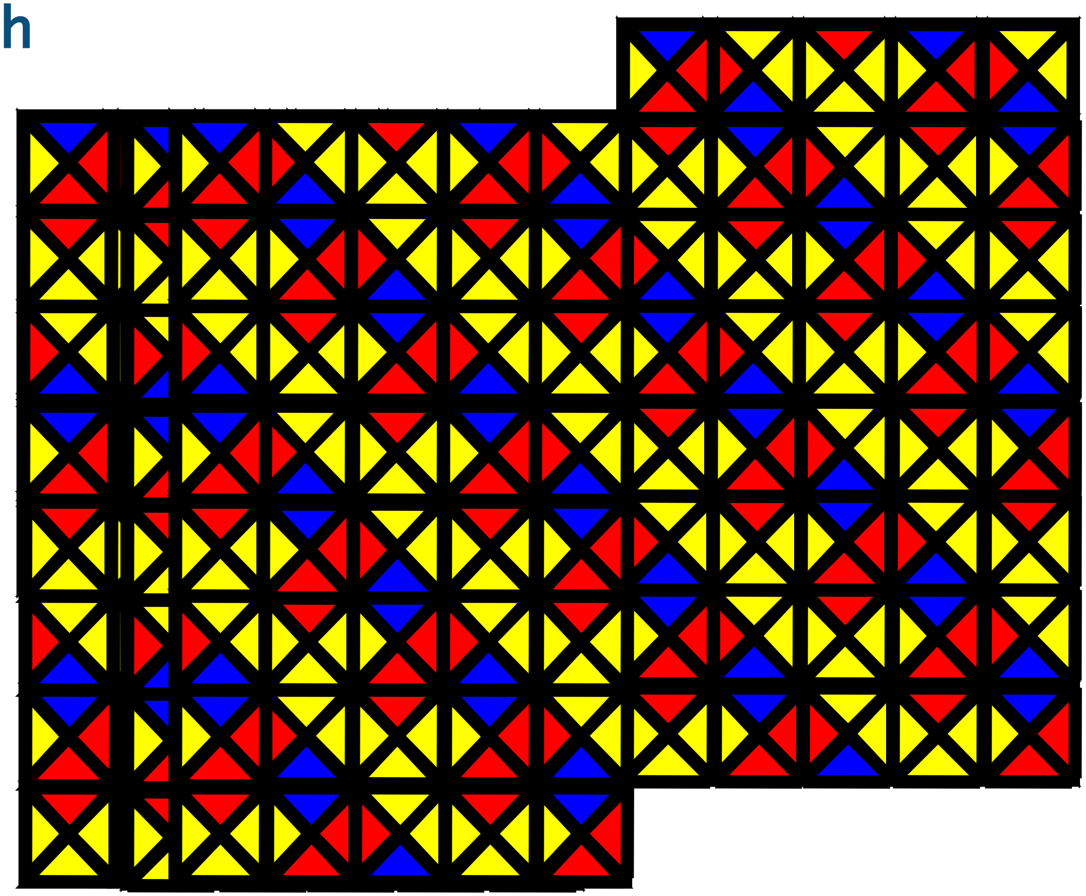
Tiling problem – Extending to a larger region

Overlap the top two rows with the bottom two rows

– obtain an 8×5 tiled area

Next place two of these 8×5 rectangles side by side

– with the right hand rectangle one row above the left hand rectangle

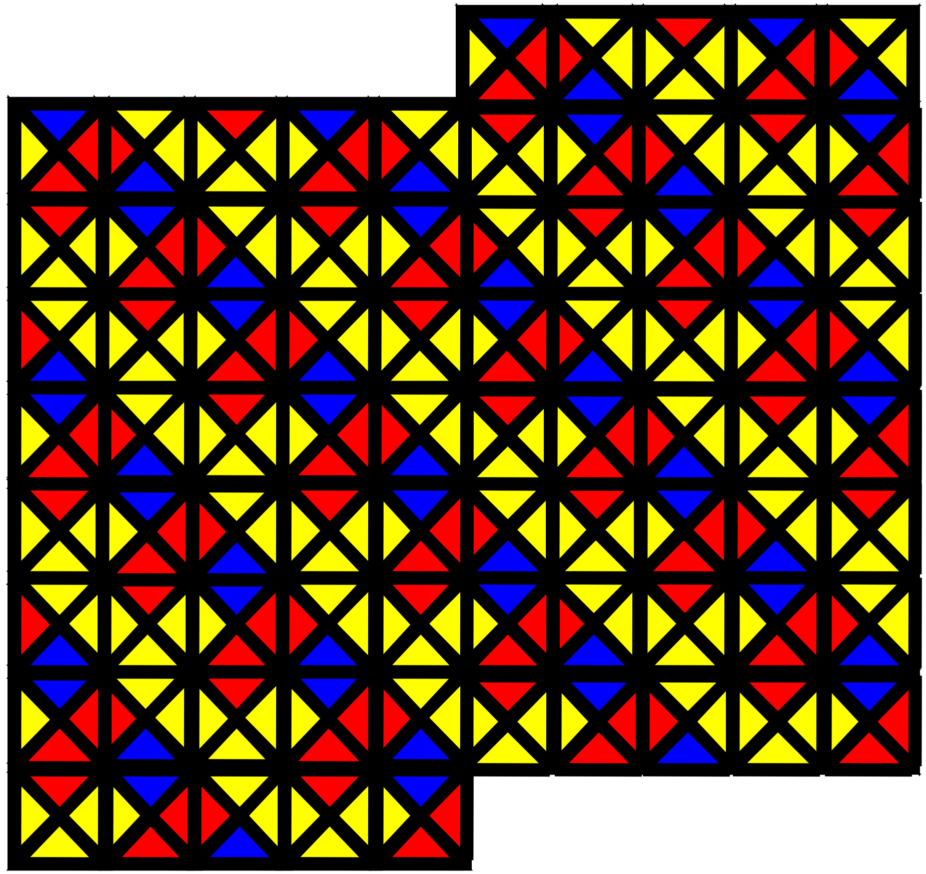


Tiling problem – Extending to a larger region

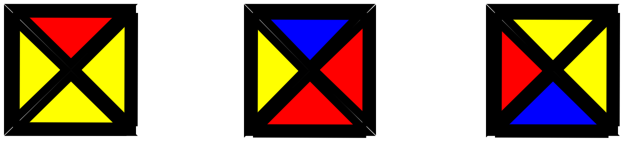
Overlap the top two rows with the bottom two rows

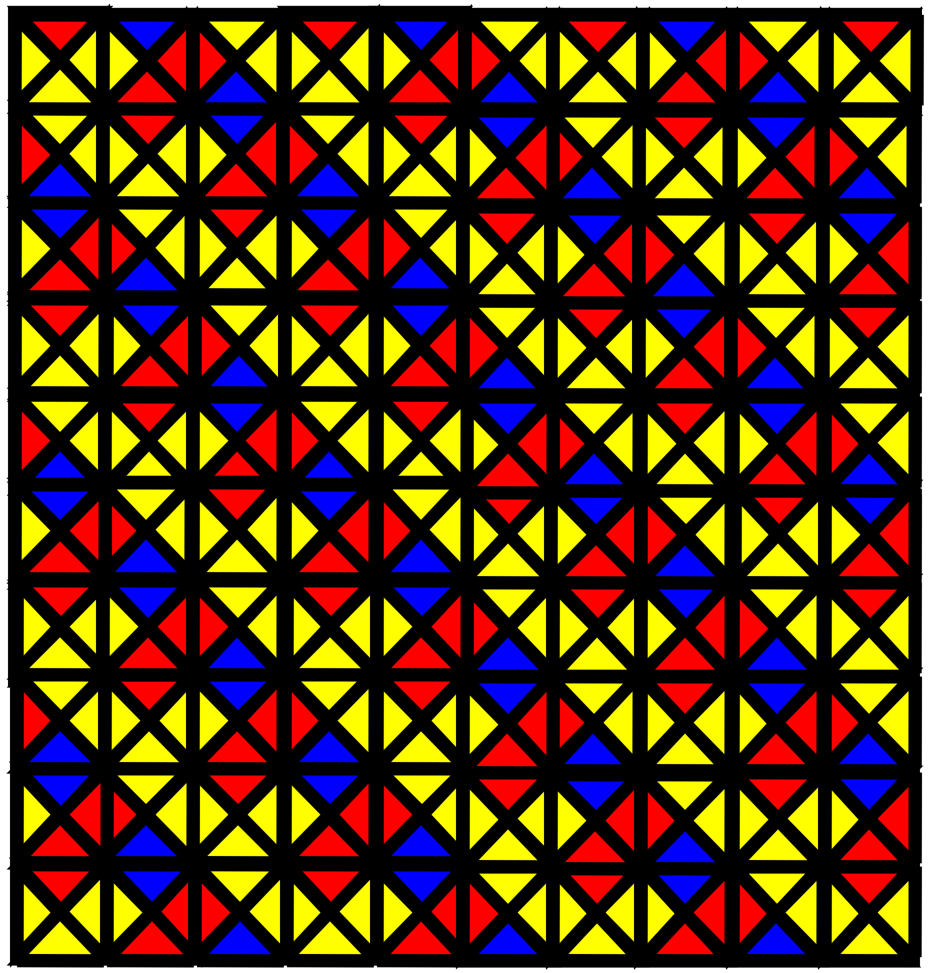
- obtain an 8×5 tiled area

By repeating this pattern it follows that any finite area can be tiled

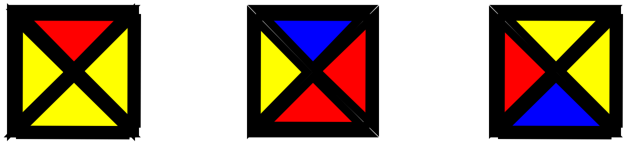


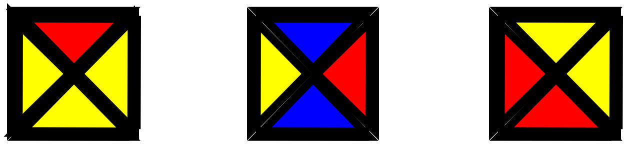
Tiling problem – Tiling a 10×10 square

Available tiles: 



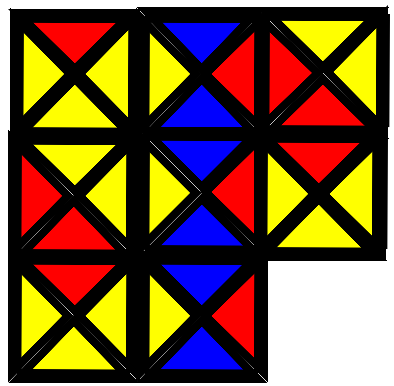
Tiling problem – Altering the tiles

Original tiles: 

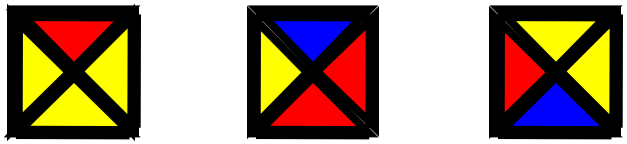
New tiles: 

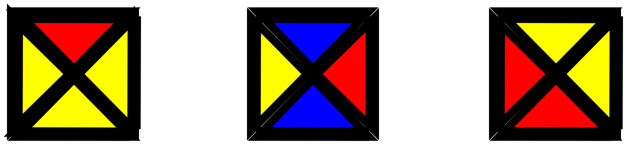
Now impossible to tile a 3×3 square

If we try:



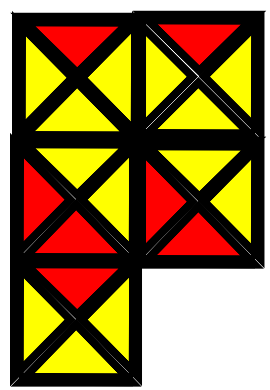
Tiling problem – Altering the tiles

Original tiles: 

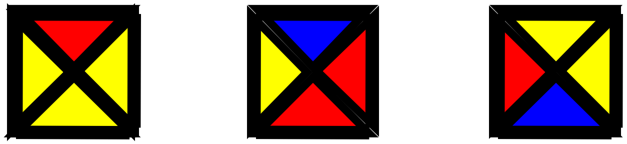
New tiles: 

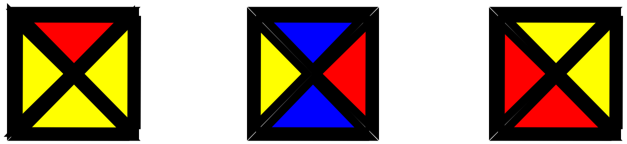
Now impossible to tile a 3×3 square

If we try:



Tiling problem – Altering the tiles

Original tiles: 

New tiles: 

Now impossible to tile a 3×3 square

There are 3^9=19,683 possibilities if you want to try them all out…

Tiling problem

Tiling problem: given a set of tile descriptions, can any finite area, of any size, be completely 'tiled' using only tiles from this set?

There is no algorithm for the tiling problem

- it's been proved that for any algorithm A that we might try to formulate there is a set of tiles S for which either A does not terminate or A gives the wrong answer

The problem is that:

- "any size" means we do have to check all finite areas and there are infinitely many of these
- and for certain sets of tile descriptions that can tile any area, there is no "repeated pattern" we can use
- so to be correct the algorithm would really have to check all finite areas

Undecidable problems

A problem Π that admits no algorithm is called **non-computable** or **unsolvable**

If Π is a decision problem and Π admits no algorithm it is called **undecidable**

The Tiling Problem is undecidable

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \ldots, i_r$ of integers chosen from $\{1, \ldots, n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{i_j}'s and the Y_{i_j}'s gives the same result

Example: $n=5$

- X_1 = abb, X_2 = a, X_3 = bab, X_4 = baba, X_5 = aba
- Y_1 = bbab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- correspondence is given by the sequence 2, 1, 1, 4, 1, 5
  - word constructed from X_i's:
  - word constructed from Y_i's:

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1,\dots,X_n$ and $Y_1,\dots,Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \dots, i_r$ of integers chosen from $\{1,\dots,n\}$ such that $X_{i1}X_{i2}\dots X_{ir} = Y_{i1}Y_{i2}\dots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: $n=5$

- X_1 = abb, X_2 = a, X_3 = bab, X_4 = baba, X_5 = aba
- Y_1 = bbab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- correspondence is given by the sequence 2, 1, 1, 4, 1, 5
  - word constructed from X_i's: a
  - word constructed from Y_i's: aa

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \ldots, i_r$ of integers chosen from $\{1, \ldots, n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: $n=5$

- $X_1 = abb$, $X_2 = a$, $X_3 = bab$, $X_4 = baba$, $X_5 = aba$
- $Y_1 = bbab$, $Y_2 = aa$, $Y_3 = ab$, $Y_4 = aa$, $Y_5 = a$
- correspondence is given by the sequence 2, **1**, 1, 4, 1, 5
  - word constructed from X_i's: aabb
  - word constructed from Y_i's: aabbab

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \ldots, i_r$ of integers chosen from $\{1, \ldots, n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: $n=5$

- X_1 = abb, X_2 = a, X_3 = bab, X_4 = baba, X_5 = aba
- Y_1 = bbab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- correspondence is given by the sequence 2, 1, **1**, 4, 1, 5
  - word constructed from X_i's: aabbabb
  - word constructed from Y_i's: aabbabbbab

Post’s correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \ldots, i_r$ of integers chosen from $\{1, \ldots, n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: $n=5$

- X_1 = abb, X_2 = a, X_3 = bab, X_4 = **baba**, X_5 = aba
- Y_1 = bbab, Y_2 = aa, Y_3 = ab, Y_4 = **aa**, Y_5 = a
- correspondence is given by the sequence 2, 1, 1, **4**, 1, 5
  - word constructed from X_i’s: aabbabbbaba
  - word constructed from Y_i’s: aabbabbbabaa

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1,\ldots,X_n$ and $Y_1,\ldots,Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \ldots, i_r$ of integers chosen from $\{1,\ldots,n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: $n=5$

- X_1 = **abb**, X_2 = a, X_3 = bab, X_4 = baba, X_5 = aba
- Y_1 = **bbab**, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- correspondence is given by the sequence 2, 1, 1, 4, **1**, 5
  - word constructed from X_i's: aabbabbbaba**abb**
  - word constructed from Y_i's: aabbabbbaba**bbab**

Post’s correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \ldots, i_r$ of integers chosen from $\{1, \ldots, n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: $n=5$

- X_1 = abb, X_2 = a, X_3 = bab, X_4 = baba, X_5 = **aba**
- Y_1 = bbab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = **a**
- correspondence is given by the sequence 2, 1, 1, 4, 1, **5**
  - word constructed from X_i’s: aabbabbbabaabbaba
  - word constructed from Y_i’s: aabbabbbabaabbaba

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \ldots, i_r$ of integers chosen from $\{1, \ldots, n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: $n=5$

- X_1 = abb, X_2 = a, X_3 = bab, X_4 = baba, X_5 = aba
- Y_1 = bbab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- correspondence is given by the sequence 2, 1, 1, 4, 1, 5
  - word constructed from X_i's: **aabbabbbabaabbaba**
  - word constructed from Y_i's: **aabbabbbabaabbaba**

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \ldots, i_r$ of integers chosen from $\{1, \ldots, n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: **n=5** (with first letter from X_1 and Y_1 removed)

- X_1 = bb, X_2 = a, X_3 = bab, X_4 = bab, X_5 = aba
- Y_1 = bab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1,\dots,X_n$ and $Y_1,\dots,Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \dots, i_r$ of integers chosen from $\{1,\dots,n\}$ such that $X_{i1}X_{i2}\dots X_{ir} = Y_{i1}Y_{i2}\dots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: **n=5** (with first letter from X_1 and Y_1 removed)

- X_1 = bb, X_2 = a, X_3 = bab, X_4 = bab, X_5 = aba
- Y_1 = bab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- to get a match we must start with either 2 or 5

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1, \dots, X_n$ and $Y_1, \dots, Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \dots, i_r$ of integers chosen from $\{1, \dots, n\}$ such that $X_{i1}X_{i2}\dots X_{ir} = Y_{i1}Y_{i2}\dots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: $n=5$ (with first letter from X_1 and Y_1 removed)

- X_1 = bb, X_2 = a, X_3 = bab, X_4 = bab, X_5 = aba
- Y_1 = bab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- suppose we start with 2
  - word constructed from X_i's: **a**
  - word constructed from Y_i's: **aa**

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1,\dots,X_n$ and $Y_1,\dots,Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \dots, i_r$ of integers chosen from $\{1,\dots,n\}$ such that $X_{i1}X_{i2}\dots X_{ir} = Y_{i1}Y_{i2}\dots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: **n=5** (with first letter from X_1 and Y_1 removed)

- X_1 = bb, X_2 = a, X_3 = bab, X_4 = bab, X_5 = aba
- Y_1 = bab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- but then must repeatedly follow with 2 as again nothing else matches
  - word constructed from X_i's: **aa**
  - word constructed from Y_i's: **aaaa**

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1,\ldots,X_n$ and $Y_1,\ldots,Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1,i_2,\ldots,i_r$ of integers chosen from $\{1,\ldots,n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: $n=5$ (with first letter from X_1 and Y_1 removed)

- X_1 = bb, X_2 = a, X_3 = bab, X_4 = bab, X_5 = aba
- Y_1 = bab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- but then must repeatedly follow with 2 as again nothing else matches
  - word constructed from X_i's: **aaa**
  - word constructed from Y_i's: **aaaaaa**

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1,\ldots,X_n$ and $Y_1,\ldots,Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1,i_2,\ldots,i_r$ of integers chosen from $\{1,\ldots,n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: **n=5** (with first letter from X_1 and Y_1 removed)

- X_1 = bb, X_2 = a, X_3 = bab, X_4 = bab, X_5 = aba
- Y_1 = bab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- but then must repeatedly follow with 2 as again nothing else matches
    - word constructed from X_i's: **aaaa**
    - word constructed from Y_i's: **aaaaaaa**

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \ldots, i_r$ of integers chosen from $\{1, \ldots, n\}$ such that $X_{i1} X_{i2} \ldots X_{ir} = Y_{i1} Y_{i2} \ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: $n=5$ (with first letter from X_1 and Y_1 removed)

- X_1 = bb, X_2 = a, X_3 = bab, X_4 = bab, X_5 = aba
- Y_1 = bab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- suppose we start with 5
  - word constructed from X_i's: **aba**
  - word constructed from Y_i's: **a**

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1,\ldots,X_n$ and $Y_1,\ldots,Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1,i_2,\ldots,i_r$ of integers chosen from $\{1,\ldots,n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: **n=5** (with first letter from X_1 and Y_1 removed)

- X_1 = bb, X_2 = a, X_3 = bab, X_4 = bab, X_5 = aba
- Y_1 = bab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- suppose we start with 5 must follow with 1 to get a match
  - word constructed from X_i's: **ababb**
  - word constructed from Y_i's: **abab**

Post’s correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \ldots, i_r$ of integers chosen from $\{1, \ldots, n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: **n=5** (with first letter from X_1 and Y_1 removed)

- X_1 = bb, X_2 = a, X_3 = bab, X_4 = bab, X_5 = aba
- Y_1 = bab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- suppose we start with 5 must follow with 1, then we are stuck
  - word constructed from X_i’s: **ababb**
  - word constructed from Y_i’s: **abab**

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1,\dots,X_n$ and $Y_1,\dots,Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \dots, i_r$ of integers chosen from $\{1,\dots,n\}$ such that $X_{i1}X_{i2}\dots X_{ir} = Y_{i1}Y_{i2}\dots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Example: $n=5$ (with first letter from X_1 and Y_1 removed)

- X_1 = bb, X_2 = a, X_3 = bab, X_4 = bab, X_5 = aba
- Y_1 = bab, Y_2 = aa, Y_3 = ab, Y_4 = aa, Y_5 = a
- follows that we can now never get a correspondence

Post's correspondence problem (PCP)

A **word** is a finite string over some given finite alphabet

Instance: two finite sequences of words $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$

- the words are all over the same alphabet

Question: does there exist a sequence $i_1, i_2, \ldots, i_r$ of integers chosen from $\{1, \ldots, n\}$ such that $X_{i1}X_{i2}\ldots X_{ir} = Y_{i1}Y_{i2}\ldots Y_{ir}$?

- i.e. concatenating the X_{ij}'s and the Y_{ij}'s gives the same result

Post's Correspondence Problem is undecidable

- there is no algorithm that will either always give the right answer or always terminate

The halting problem

An impossible project: write a program Q that takes as input

- a legal program X (say in Java)
- an input string S for program X

and returns as output

- **yes** if program X halts (terminates) when run with input S
- **no** if program X enters an infinite loop (doesn't terminate) when run with input S

It has been proved that no such program Q can exists, meaning the halting problem is undecidable

The halting problem

Example (small) programs

```
public void test(int n){
  if (n == 1)
    while (true)
      null;
}
```

The program ‘test’ will terminate if and only if input $n \neq 1$

The halting problem

Example (small) programs

```
public int erratic(int n){
  while (n != 1)
    if (n % 2 == 0) n = n/2;
    else n = 3*n + 1;
}
```

For example if ‘**erratic**’ is called with **n=7** sequence of values:

22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Nobody knows whether ‘**erratic**’ terminates for all values of **n**

The halting problem – Undecidability

A formal definition of the halting problem (**HP**)

Instance: a legal Java program **X** and an input string **S** for **X**

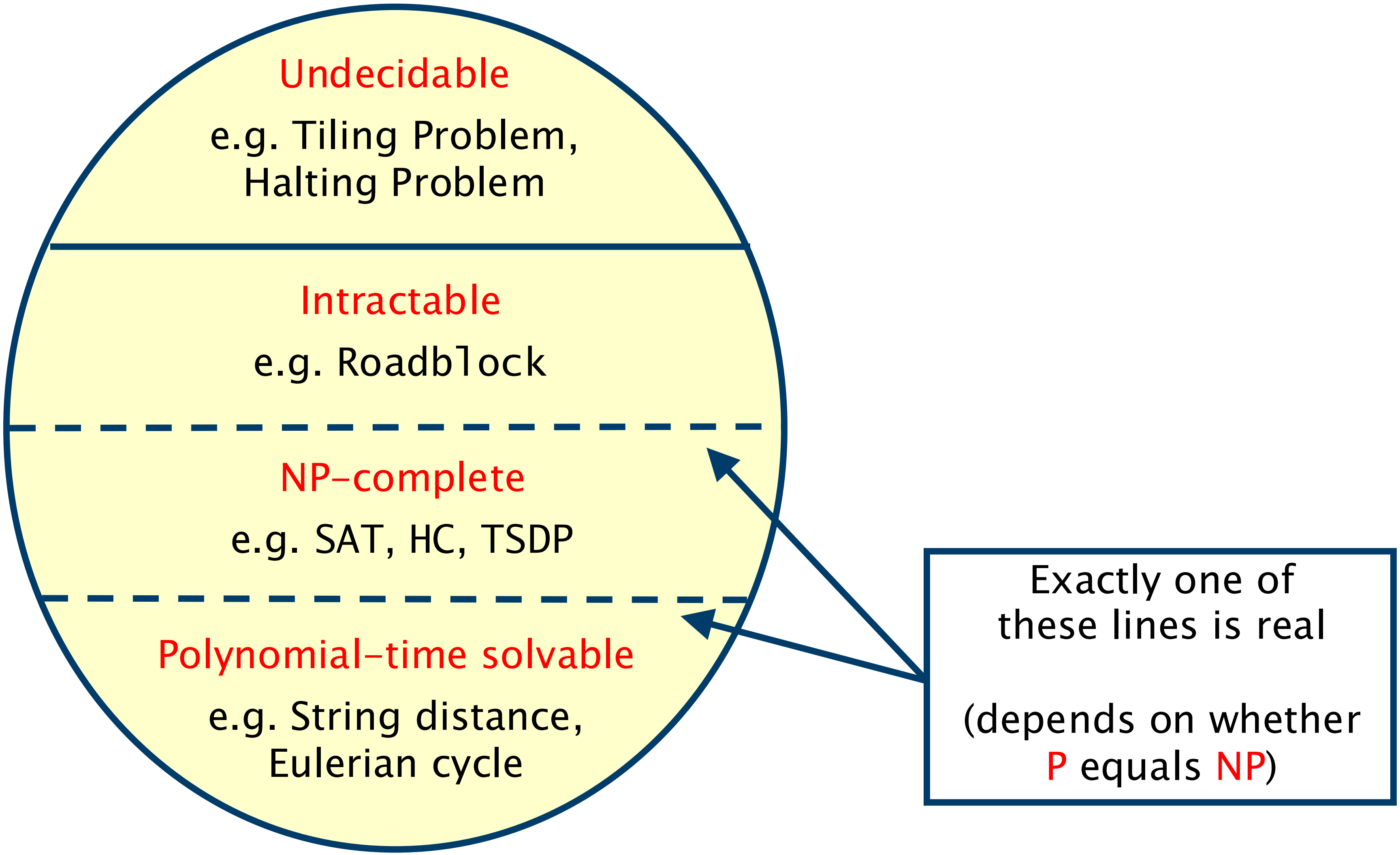
- can substitute any language for Java

Question: does **X** halt when run with input **S**?

Theorem: **HP is undecidable**

- proof by contradiction – at the end of these slide notes
- pre-recorded video available on Moodle
- this proof is supplementary material – non examinable

Hierarchy of decision problems



Section 5 – Computability

Introduction

The halting problem

Models of computation

- finite-state automata
- pushdown automata
- Turing machines
- Counter machines
- Church–Turing thesis

Models of computation

input x → black box → output f(x)

Attempts to define "the black box"

– we will look at three classical models of computation of increasing power

- **Finite-State Automata**
  - simple machines with a fixed amount of memory
  - have very limited (but still useful) problem-solving ability
- **Pushdown Automata (PDA)**
  - simple machines with an unlimited memory that behaves like a stack
- **Turing machines (TM)**
  - simple machines with an unlimited memory that can be used essentially arbitrarily, in any way
  - these have essentially the same power as a typical computer

Section 5 – Computability

Introduction

Models of computation

- **finite-state automata**
- pushdown automata
- Turing machines
- Counter machines
- Church–Turing thesis

Deterministic finite-state automata

Simple machines with limited memory which **recognise** input on a read-only tape

A DFA consists of

- a finite input alphabet Σ
- a finite set of states Q
- a initial/start state $q_0 \in Q$ and set of accepting states $F \subseteq Q$
- control/program or transition relation $T \subseteq (Q \times \Sigma) \times Q$
  - $((q,a),q') \in T$ means if in state q and read a, then move to state q'

- deterministic means that if

  $((q,a_1),q_1),\ ((q,a_2),q_2) \in T$ either $a_1 \neq a_2$ or $q_1 = q_2$
- i.e. for any state and action there is at most one move (i.e. no choice)

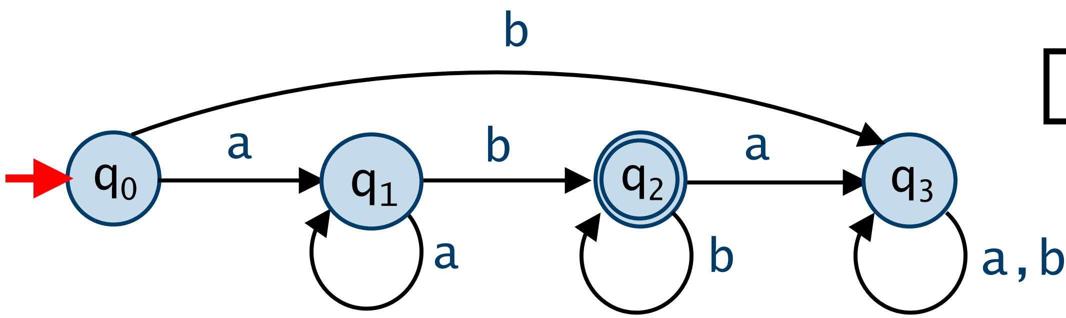
Deterministic finite-state automata

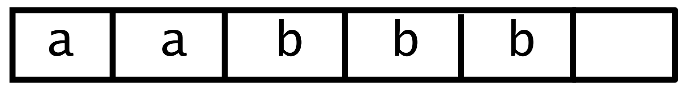
Simple machines with limited memory which **recognise** input on a read-only tape

A DFA consists of

- a finite input alphabet Σ
- a finite set of states Q
- a initial/start state $q_0 \in Q$ and set of accepting states $F \subseteq Q$
- control/program or transition relation $T \subseteq (Q \times \Sigma) \times Q$

add input tape (finite sequence of elements/actions from the alphabet)





initial state denoted by incomming arrow

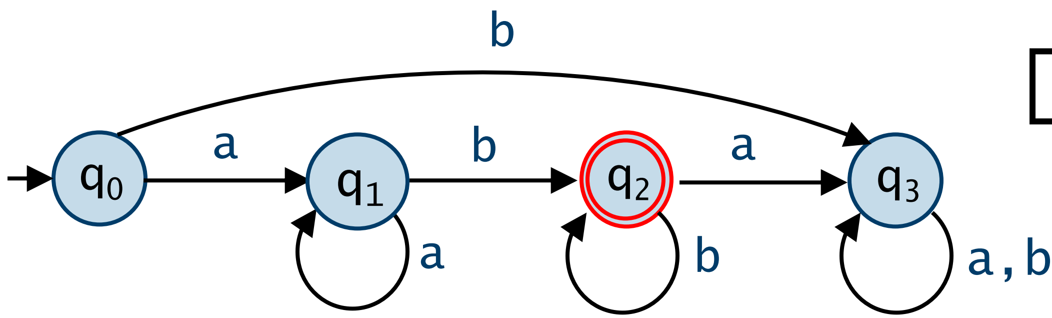
Deterministic finite-state automata

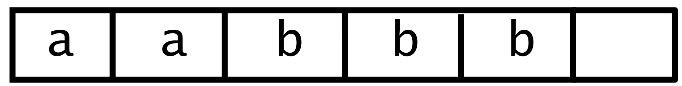
Simple machines with limited memory which recognise input on a read-only tape

A DFA consists of

- a finite input alphabet Σ
- a finite set of states Q
- a initial/start state $q_0 \in Q$ and set of accepting states $F \subseteq Q$
- control/program or transition relation $T \subseteq (Q \times \Sigma) \times Q$

add input tape (finite sequence of elements/actions from the alphabet)





accepting states denoted by double circles

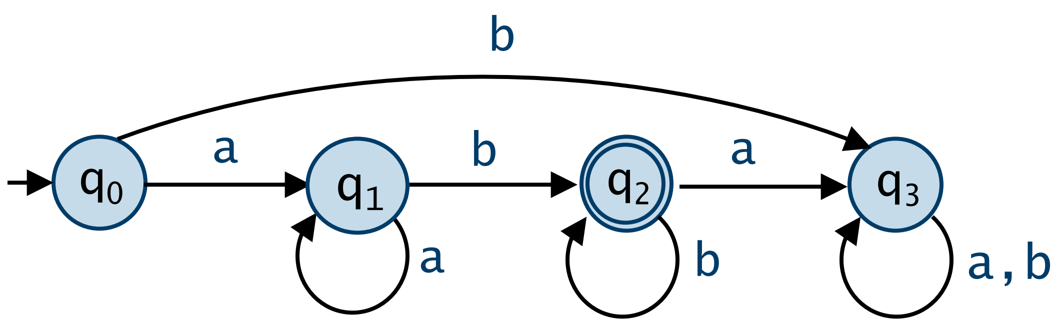
Deterministic finite-state automata

Simple machines with limited memory which recognise input on a read-only tape

A DFA consists of

- a finite input alphabet Σ
- a finite set of states Q
- a initial/start state $q_0 \in Q$ and set of accepting states $F \subseteq Q$
- control/program or transition relation $T \subseteq (Q \times \Sigma) \times Q$

add input tape (finite sequence of elements/actions from the alphabet)



control/program

((q0,a), q1)
((q0,b), q3)
((q1,a), q1)
((q1,b), q2)
((q2,a), q3)
((q2,b), q2)
((q3,a), q3)
((q3,b), q3)

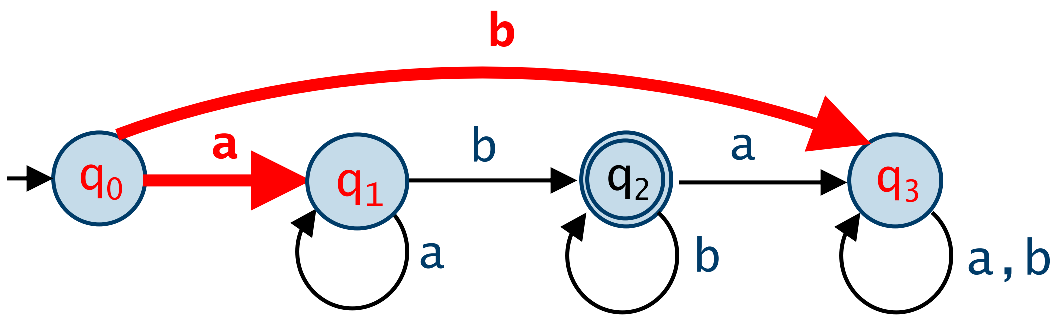
Deterministic finite-state automata

Simple machines with limited memory which recognise input on a read-only tape

add input tape (finite sequence of elements/actions from the alphabet)

A DFA consists of

- a finite input alphabet Σ
- a finite set of states Q
- a initial/start state $q_0 \in Q$ and set of accepting states $F \subseteq Q$
- control/program or transition relation $T \subseteq (Q \times \Sigma) \times Q$



control/program

((q0,a), q1)
((q0,b), q3)
((q1,a), q1)
((q1,b), q2)
((q2,a), q3)
((q2,b), q2)
((q3,a), q3)
((q3,b), q3)

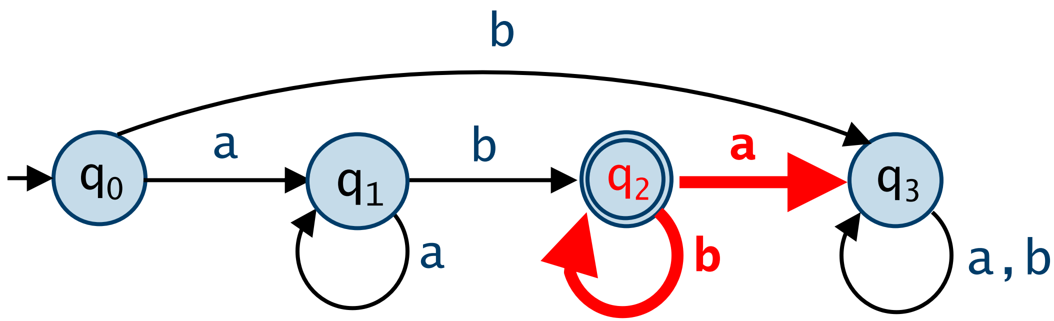
Deterministic finite-state automata

Simple machines with limited memory which recognise input on a read-only tape

A DFA consists of

- a finite input alphabet Σ
- a finite set of states Q
- a initial/start state $q_0 \in Q$ and set of accepting states $F \subseteq Q$
- control/program or transition relation $T \subseteq (Q \times \Sigma) \times Q$

add input tape (finite sequence of elements/actions from the alphabet)



control/program

((q0,a), q1)
((q0,b), q3)
((q1,a), q1)
((q1,b), q2)
((q2,a), q3)
((q2,b), q2)
((q3,a), q3)
((q3,b), q3)

Deterministic finite-state automata

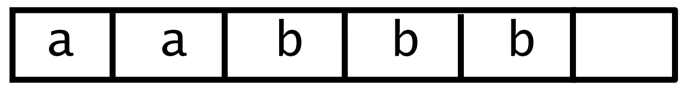
A DFA defines a language

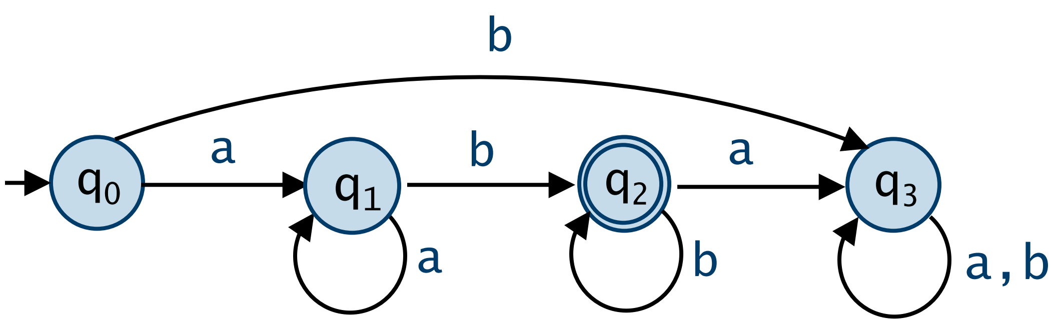
- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when 'run' end in an accepting state

Question: what language does this DFA recognise?





Deterministic finite-state automata

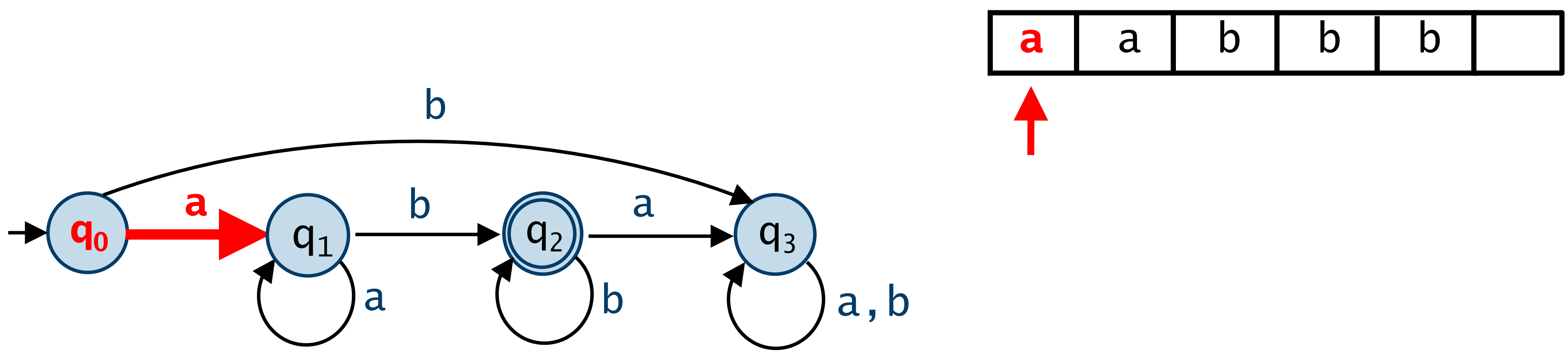
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when 'run' end in an accepting state

Question: what language does this DFA recognise?



Deterministic finite-state automata

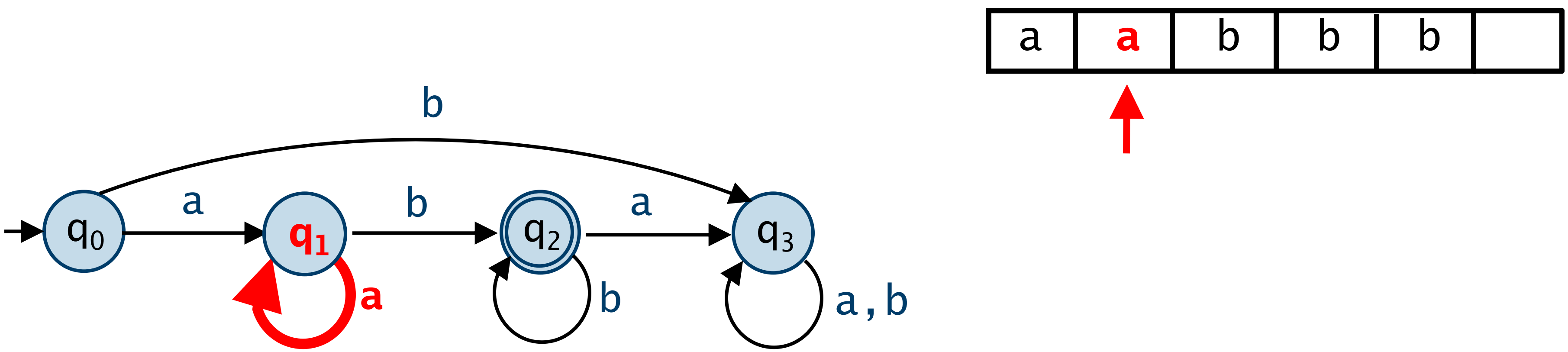
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when ‘run’ end in an accepting state

Question: what language does this DFA recognise?



Deterministic finite-state automata

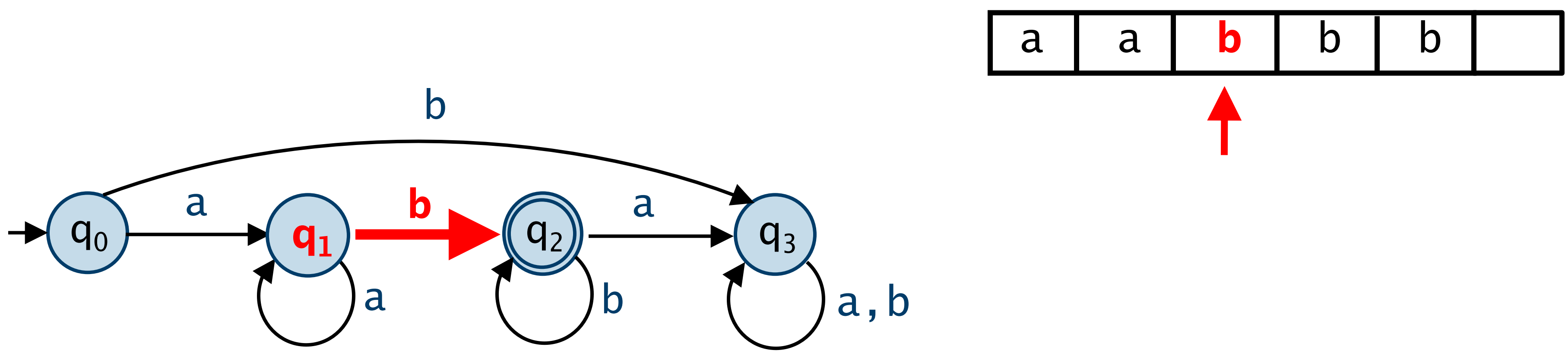
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when ‘run’ end in an accepting state

Question: what language does this DFA recognise?



Deterministic finite-state automata

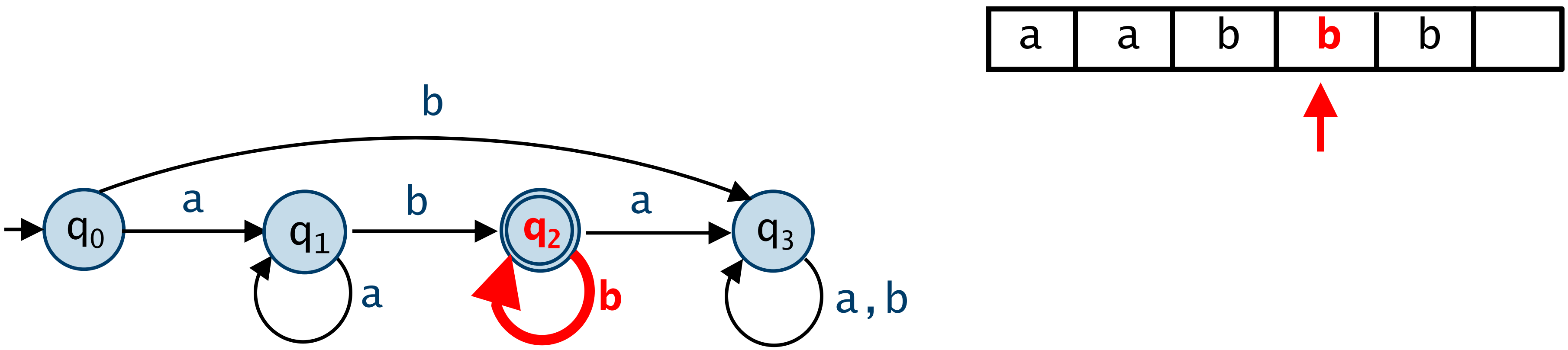
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when 'run' end in an accepting state

Question: what language does this DFA recognise?



Deterministic finite-state automata

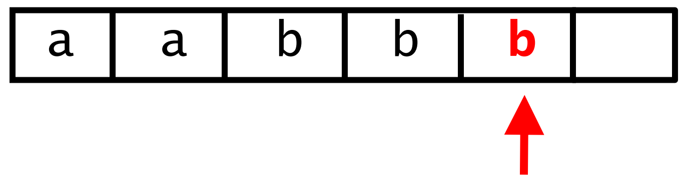
A DFA defines a language

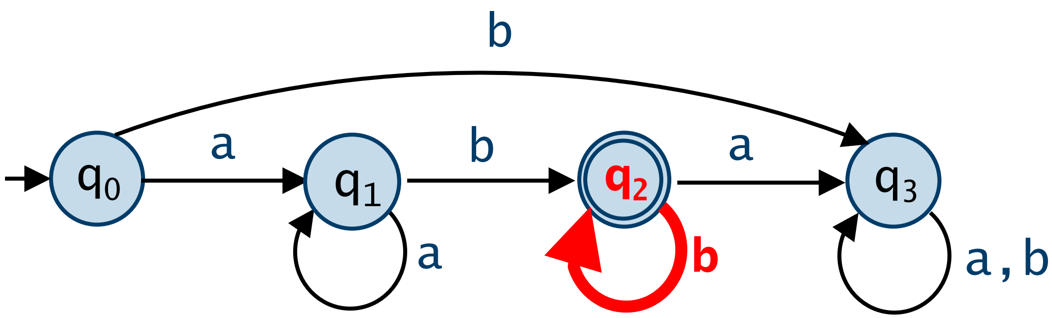
- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when ‘run’ end in an accepting state

Question: what language does this DFA recognise?





Deterministic finite-state automata

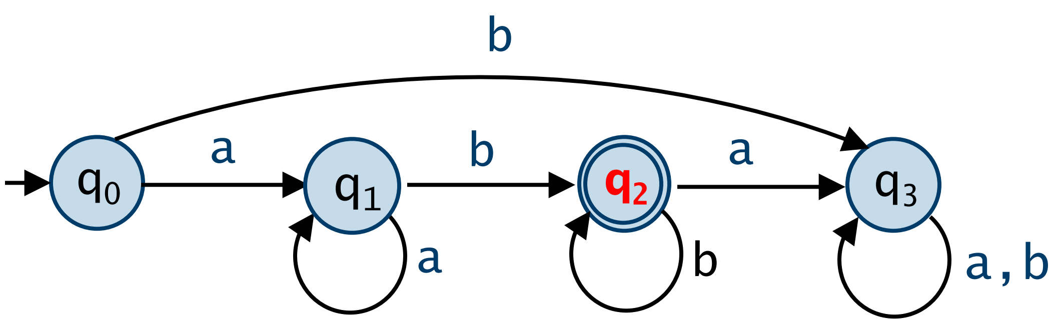
A DFA defines a language

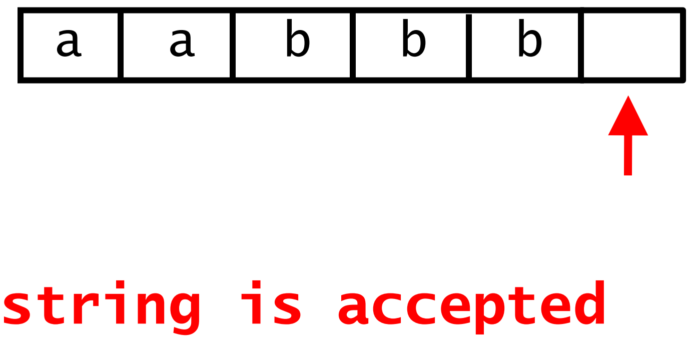
- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when ‘run’ end in an accepting state

Question: what language does this DFA recognise?





Deterministic finite-state automata

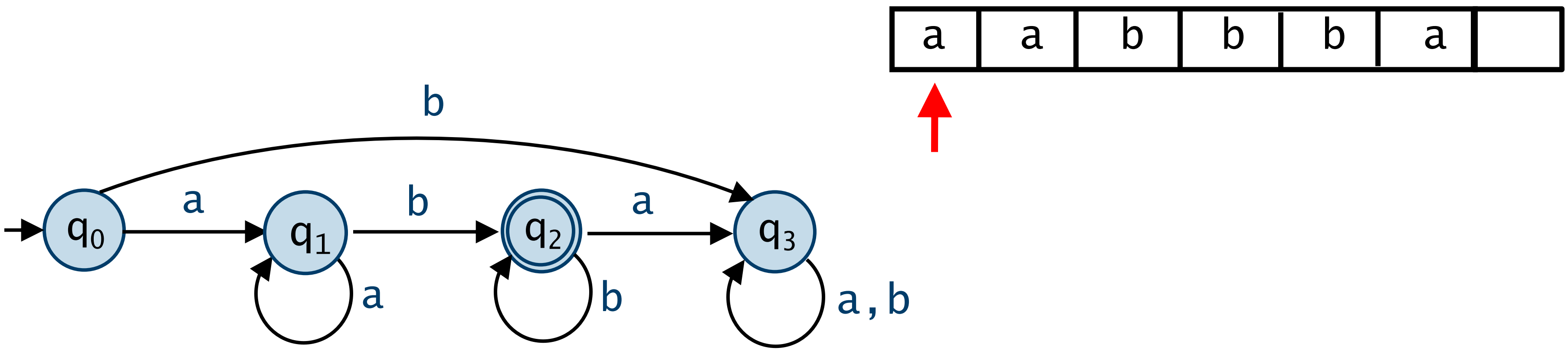
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when 'run' end in an accepting state

Question: what language does this DFA recognise?



Deterministic finite-state automata

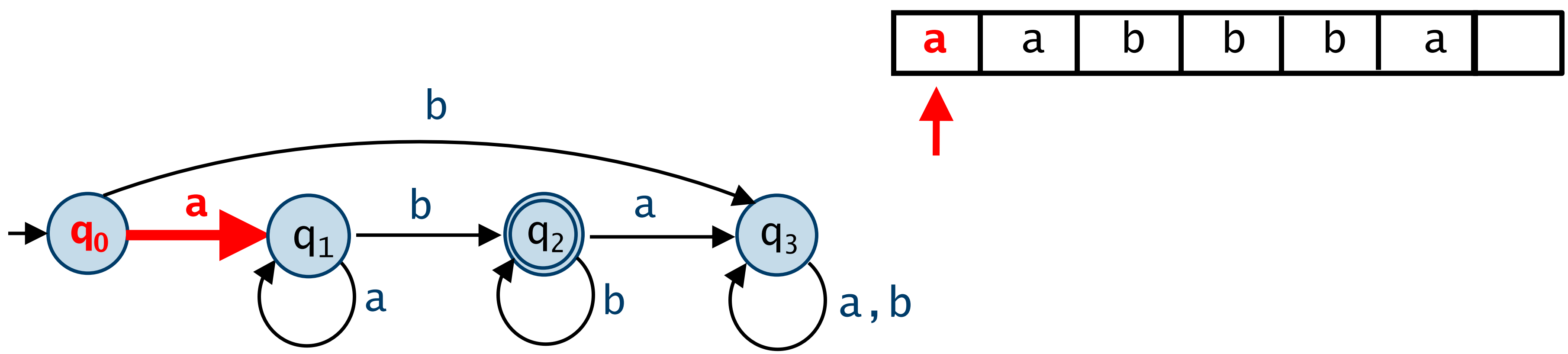
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when 'run' end in an accepting state

Question: what language does this DFA recognise?



Deterministic finite-state automata

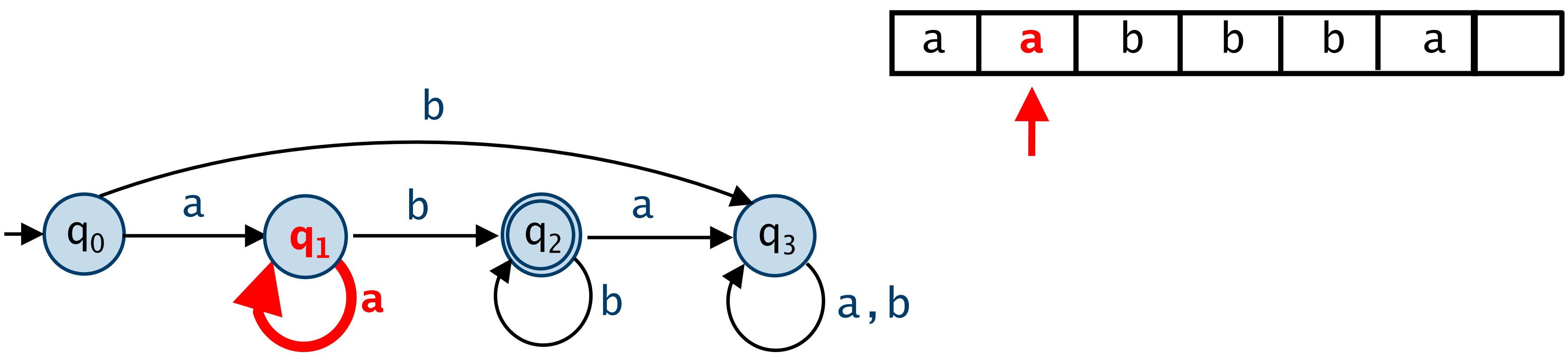
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when 'run' end in an accepting state

Question: what language does this DFA recognise?



Deterministic finite-state automata

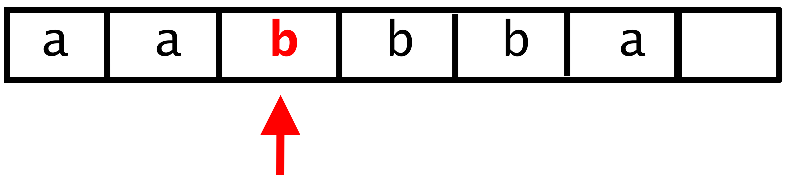
A DFA defines a language

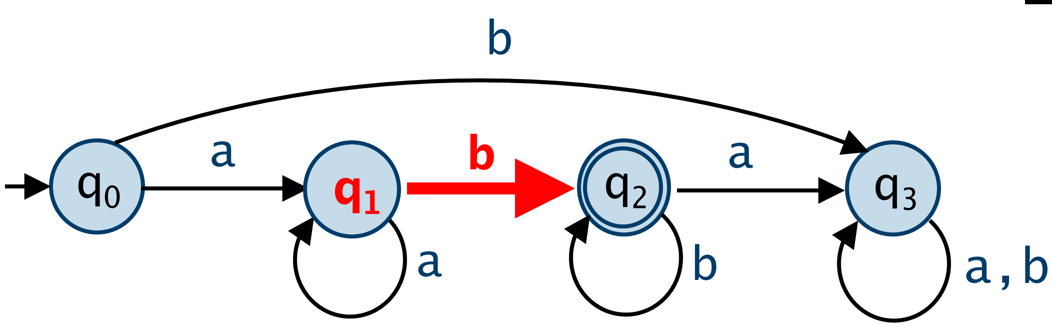
- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when ‘run’ end in an accepting state

Question: what language does this DFA recognise?





Deterministic finite-state automata

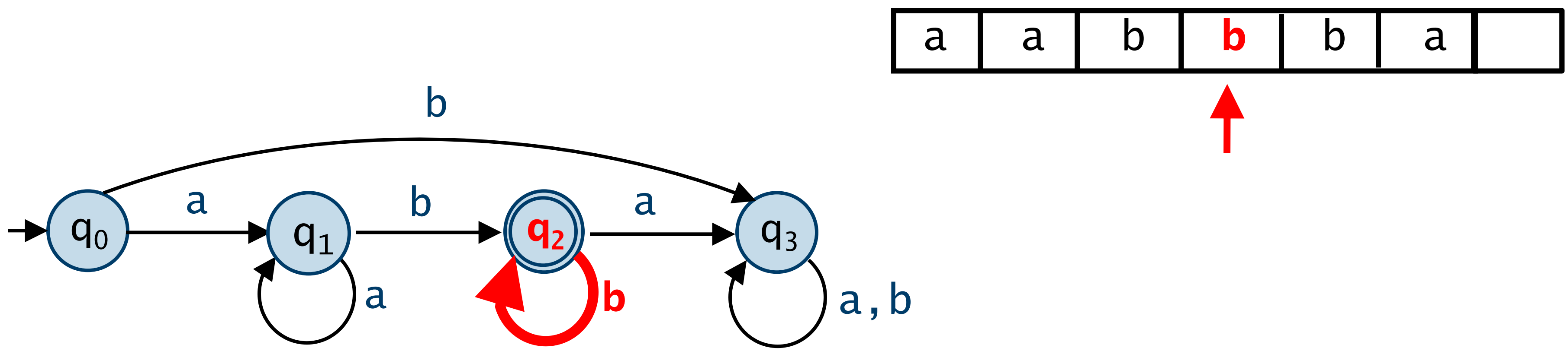
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when 'run' end in an accepting state

Question: what language does this DFA recognise?



Deterministic finite-state automata

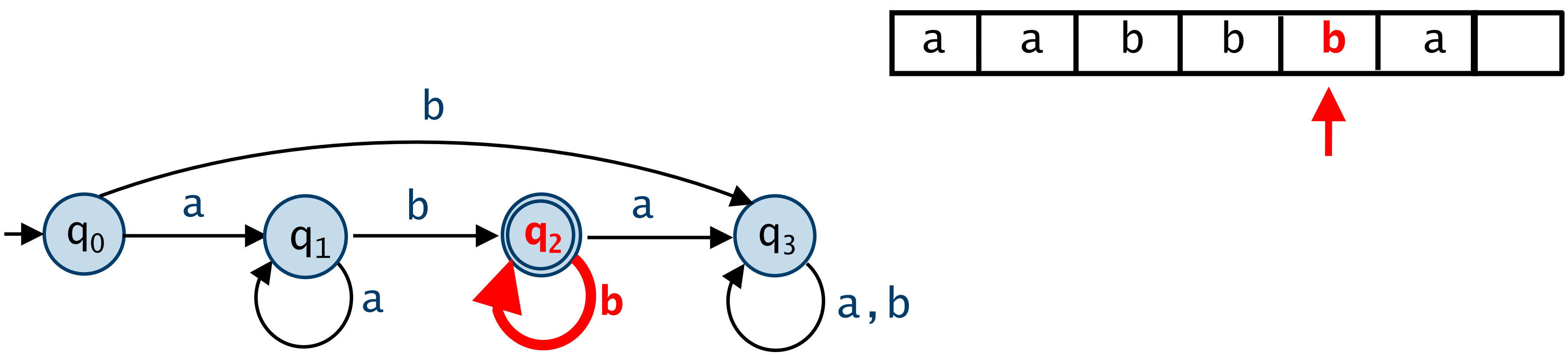
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when 'run' end in an accepting state

Question: what language does this DFA recognise?



Deterministic finite-state automata

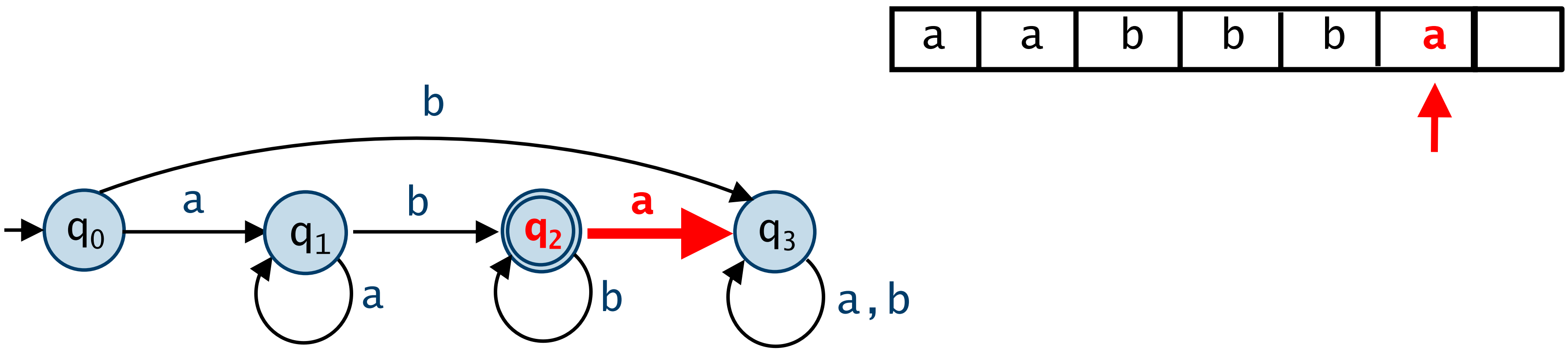
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when 'run' end in an accepting state

Question: what language does this DFA recognise?



Deterministic finite-state automata

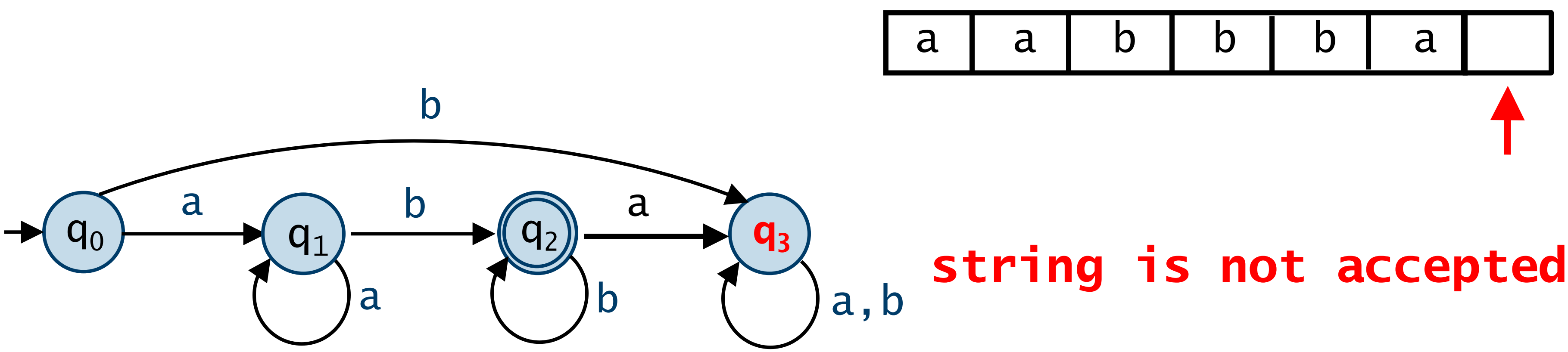
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when 'run' end in an accepting state

Question: what language does this DFA recognise?



Deterministic finite-state automata

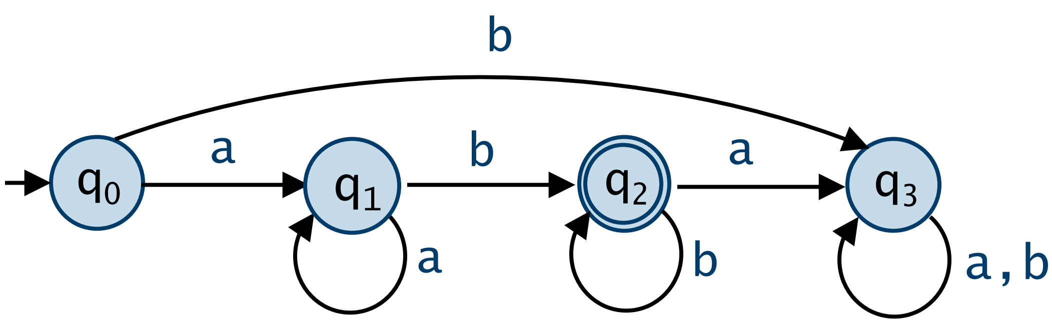
A DFA defines a language

- determines whether the string on the input tape belongs to that language
- in other words, it solves a decision problem

More precisely a DFA recognises or accepts a language

- the input strings which when 'run' end in an accepting state

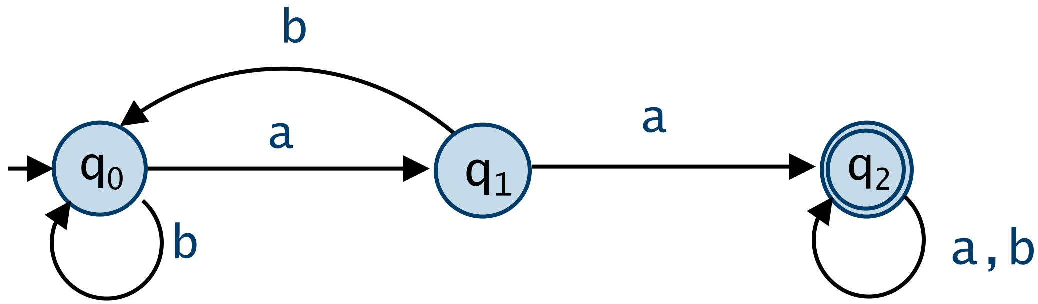
Question: what language does this DFA recognise?



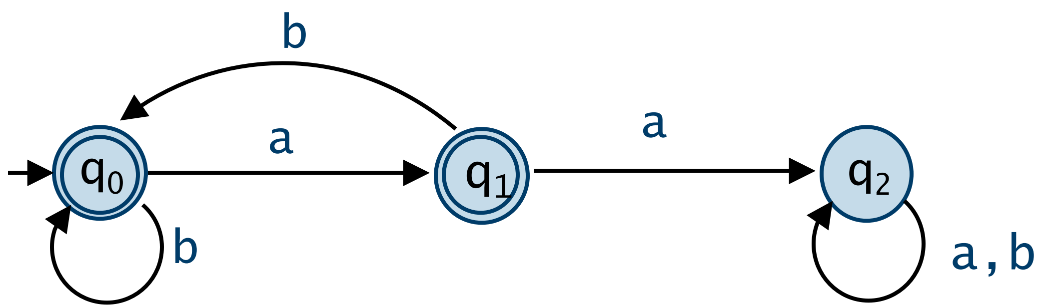
answer: the language consisting of the set of all strings comprising one or more a's followed by one or more b's

Deterministic finite-state automata

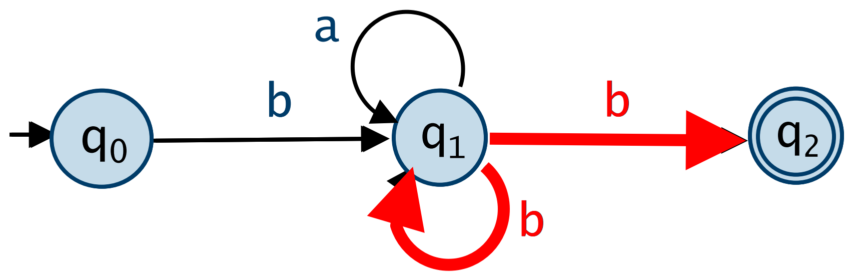
Recognises the language of strings containing two consecutive **a**'s



Recognises the complement, i.e., the language of strings that do not contain two consecutive **a**'s



Another example



Recognises strings that start and end with b

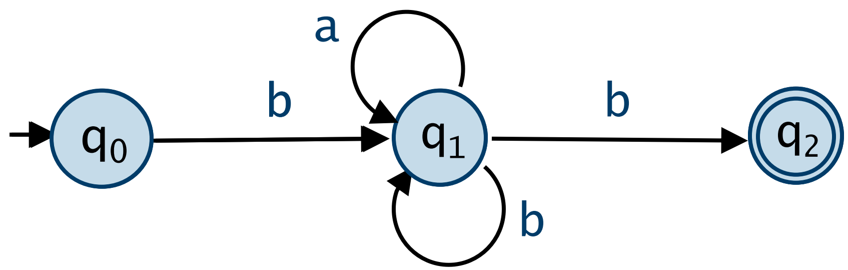
However this is not a DFA, but a non-deterministic finite-state automaton (NFA)

- in state q_1 under b can move to q_1 or q_2

Recognition for NFA is similar to non-deterministic algorithms "solving" a decision problem

- only require there exists a 'run' that ends in an accepting state
- i.e. under one possible resolution of the nondeterministic choices the input is accepted

Another example



Recognises strings that start and end with b

However this is not a DFA, but a non-deterministic finite-state automaton (NFA)

- in state q_1 under b can move to q_1 or q_2

But any NFA can be converted into a DFA

Therefore non-determinism does not expand the class of languages that can be recognised by finite state automata

- being able to guess does not give us any extra power

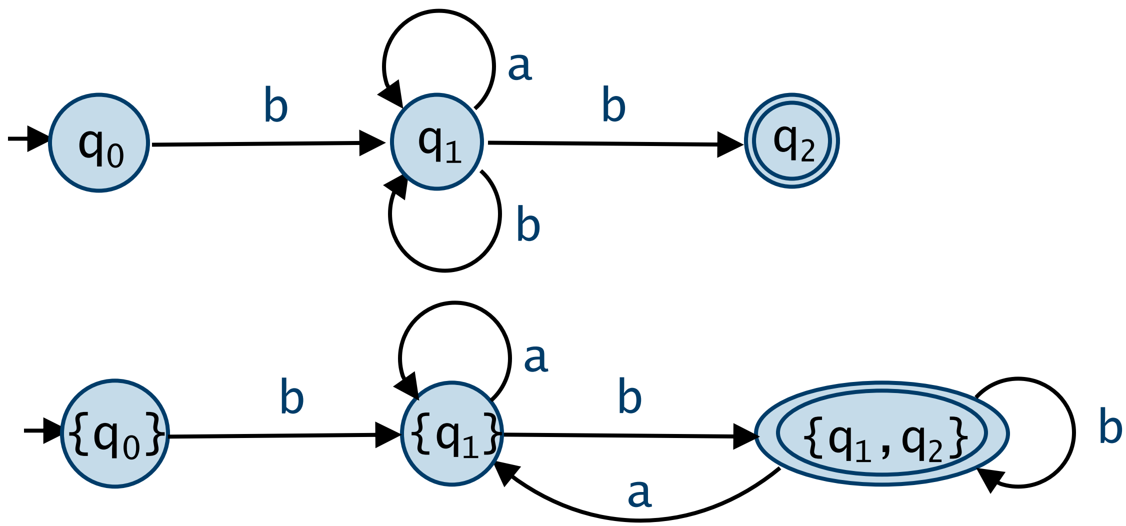
NFA to DFA reduction

Can reduce a **NFA** to a **DFA** using the subset construction

- states of the DFA are sets of states of the NFA
- construction can cause a blow-up in the number of states
  - in the worst case from N states to 2^N states

Example (without blow-up)

- recognises strings that start and end with b
- NFA
- DFA



Next time – Section 5 – Computability

Introduction

Models of computation

- finite-state automata – regular languages and regular expressions
- pushdown automata
- Turing machines
- Counter machines
- Church-Turing thesis

The halting problem

An impossible project: write a program Q that takes as input

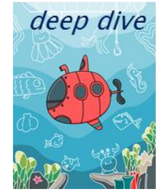
- a legal program X (say in Java)
- an input string S for program X

and returns as output

- **yes** if program X halts (terminates) when run with input S
- **no** if program X enters an infinite loop (doesn't terminate) when run with input S

It has been proved that no such program Q can exists, meaning the halting problem is undecidable

The halting problem



Example (small) programs

```
public void test(int n){
  if (n == 1)
    while (true)
      null;
}
```

The program ‘`test`’ will terminate if and only if input $n \neq 1$

The halting problem

Example (small) programs

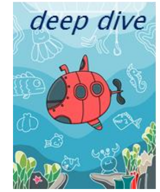
```
public int erratic(int n){
  while (n != 1)
    if (n % 2 == 0) n = n/2;
    else n = 3*n + 1;
}
```

For example if ‘**erratic**’ is called with **n=7** sequence of values:

22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Nobody knows whether ‘**erratic**’ terminates for all values of **n**

The halting problem – Undecidability



A formal definition of the halting problem (HP)

Instance: a legal Java program X and an input string S for X

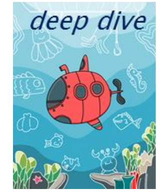
- can substitute any language for Java

Question: does X halt when run with input S?

Theorem: HP is undecidable

- proof by contradiction in the following slides
- this is non examinable

Proving undecidability by reduction



Suppose we can reduce any instance I of Π_1 into an instance J of Π_2 such that

- I has a '**yes**'-answer for Π_1 if and only if J has a "**yes**"-answer for Π_2 (like PTRs but no need for J to be constructed in polynomial time, just need to be able to construct J)

If Π_1 is undecidable and we can perform such a reduction, then Π_2 is undecidable

- proof by contradiction
- suppose Π_2 is decidable
- then using this reduction we can decide Π_1
- this is a contradiction since Π_1 is undecidable, therefore Π_2 cannot be decidable

deep dive

The halting problem – Undecidability

A formal definition of the halting problem (HP)

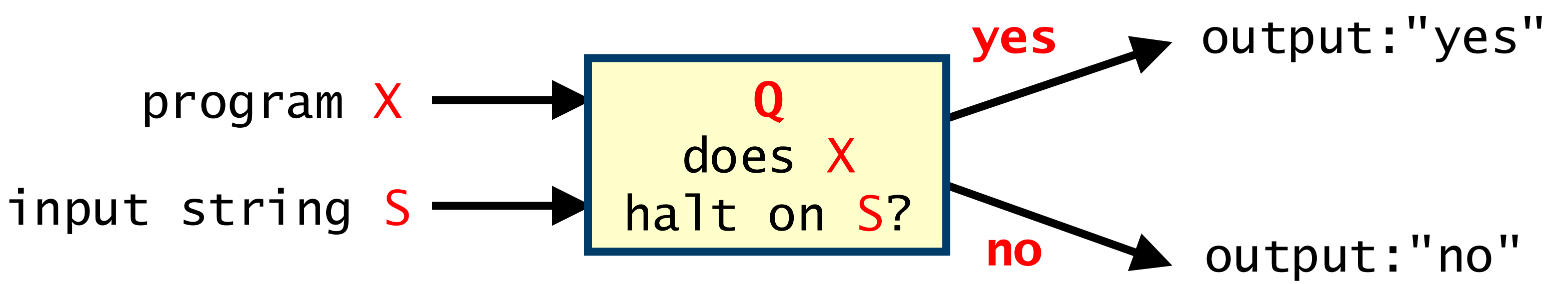
Instance: a legal Java program X and an input string S for X

- can substitute any language for Java

Question: does X halt when run with input S?

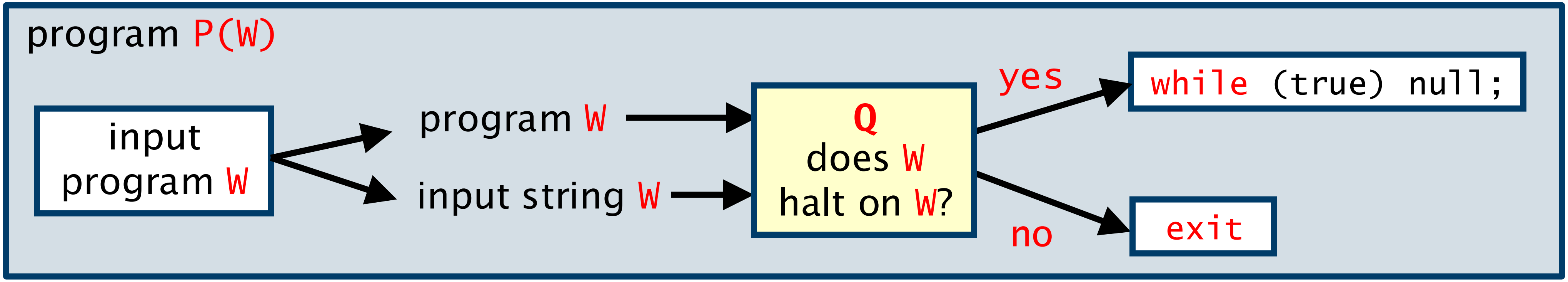
Theorem: HP is undecidable proof (by contradiction):

- suppose we have an algorithm A that decides (solves) HP
- let **Q** be an implementation of this algorithm as a Java method with X and S as parameters



The halting problem – Undecidability

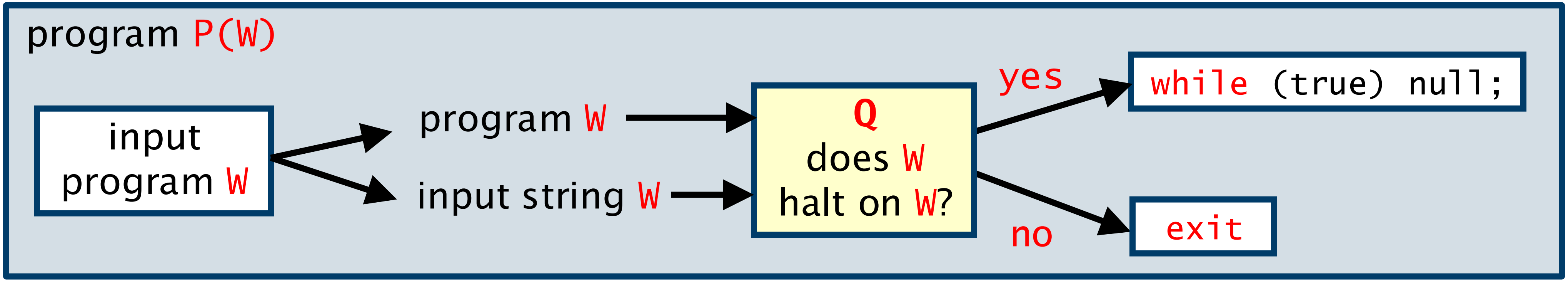
Define a new program P with input a legal program W in Java



- P makes a copy of W and calls Q(W,W)
- Q terminates by assumption, returning either "**yes**" or "**no**"
- if Q returns "**yes**", then P enters an infinite loop
- if Q returns ”**no**", then P terminates

The halting problem – Undecidability

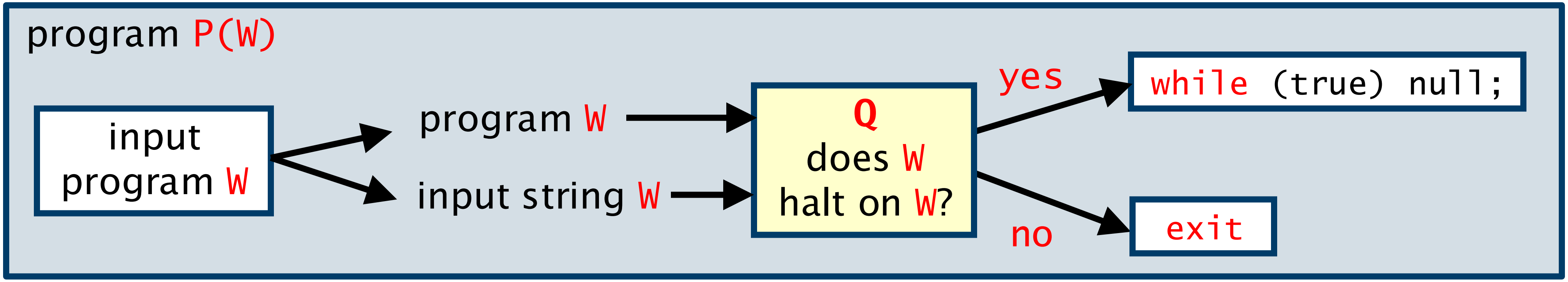
Define a new program **P** with input a legal program **W** in Java



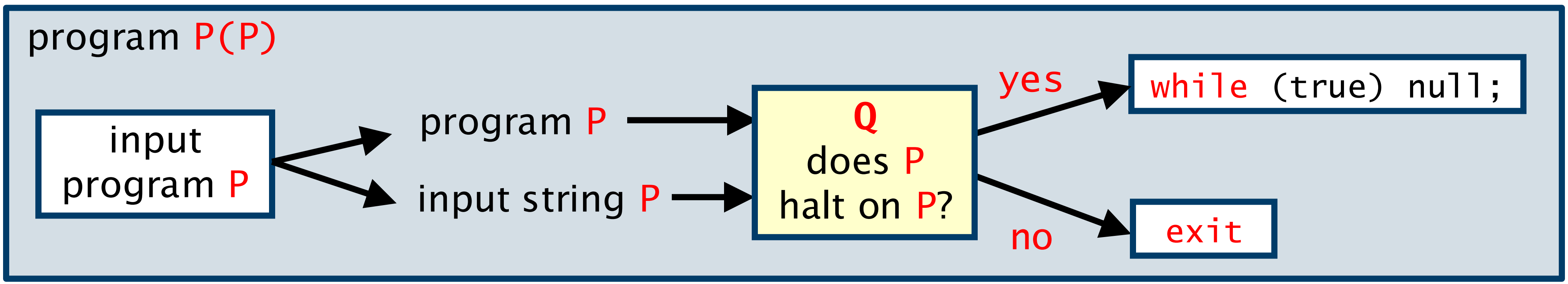
Now let the input **W** be the program **P** itself

The halting problem – Undecidability

Define a new program **P** with input a legal program **W** in Java

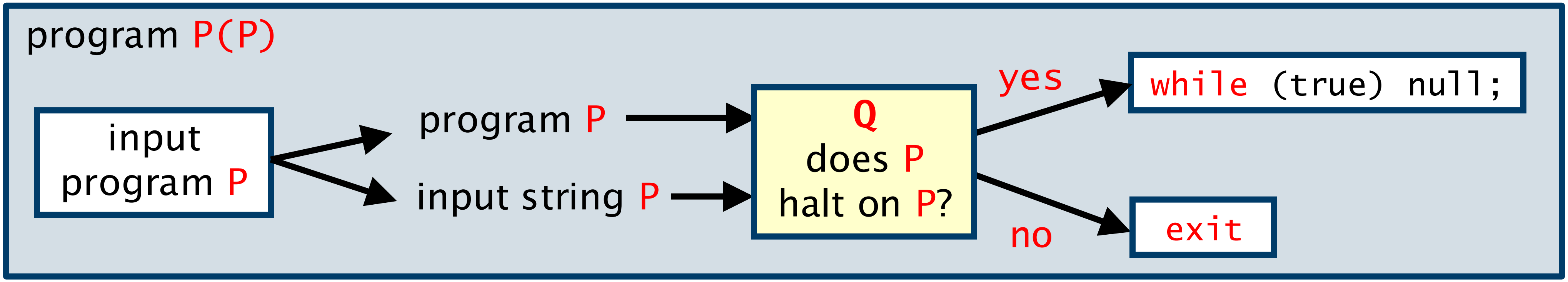


Now let the input **W** be the program **P** itself



The halting problem – Undecidability

Now let the input W to P be the program P itself

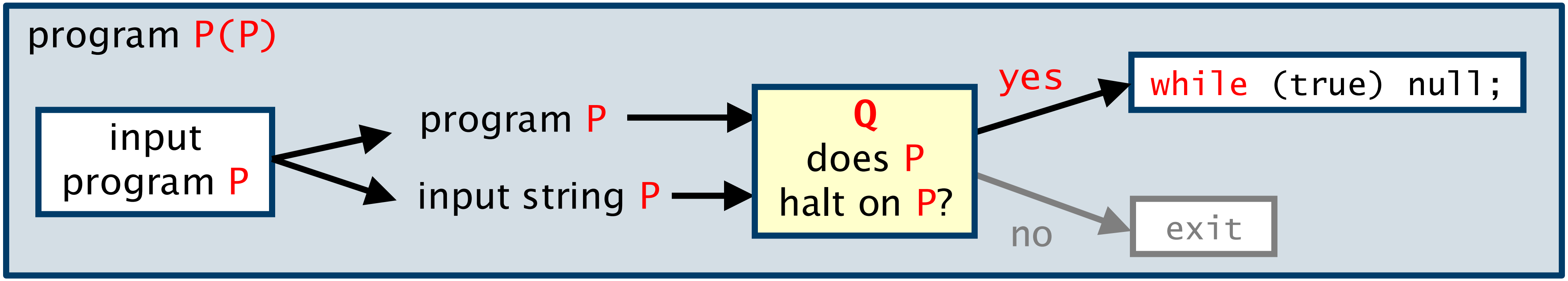


P calls Q(P,P)

- Q terminates by assumption, returning either "**yes**" or "**no**"
- recall we have assumed Q solves the halting problem

The halting problem – Undecidability

Now let the input **W** to **P** be the program **P** itself

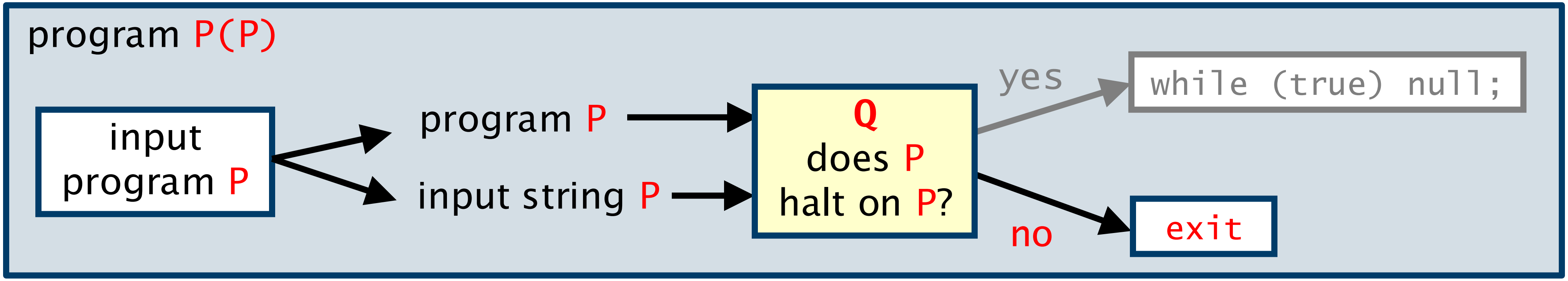


P calls Q(P,P)

- **Q** terminates by assumption, returning either "**yes**" or "**no**"
- recall we have assumed **Q** solves the halting problem
- suppose **Q** returns "**yes**", then by definition of **Q** this means P terminates
- but this also means P does not terminate (it enters the infinite loop)
  - this is a contradiction: P terminates and P does not terminate!
- therefore **Q** can't return "**yes**" and must return "**no**"

The halting problem – Undecidability

Now let the input **W** to **P** be the program **P** itself

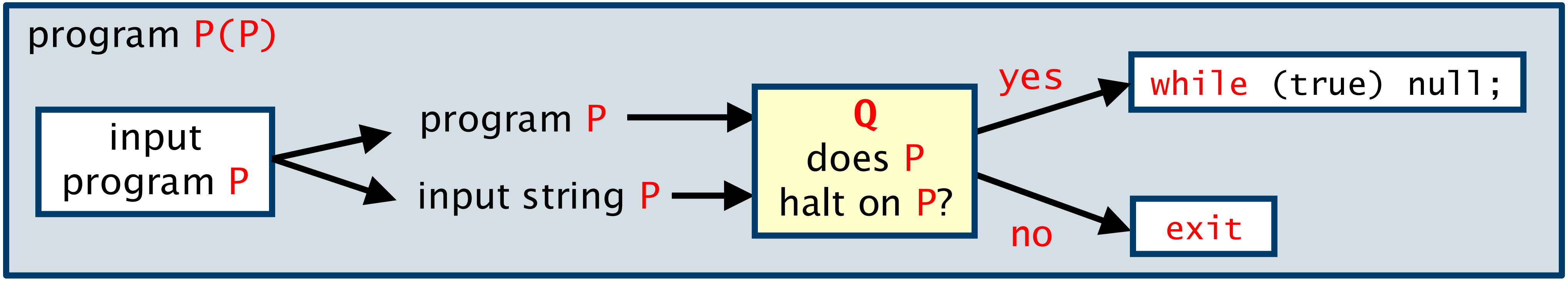


P calls Q(P,P)

- **Q** terminates by assumption, returning either "**yes**" or "**no**"
- recall we have assumed **Q** solves the halting problem
- therefore **Q** must return "**no**"
- this means by definition of **Q** that P does not terminate
- but this also means P does terminate by construction with an `exit`
- so again a contradiction

The halting problem – Undecidability

Now let the input W to P be the program P itself



P calls Q(P,P)

- Q terminates by assumption, returning either "**yes**" or "**no**"
- recall we have assumed Q solves the halting problem
- therefore Q can return neither "**yes**" nor "**no**"
- meaning no such program Q can exist
- if no such Q can exist, then no algorithm can solve the halting problem
- hence the problem is undecidable

The halting problem – Undecidability

To summarise the proof

- we assumed the existence of an algorithm A that solved HP
- implemented this algorithm as the program **Q**
- then constructed a program P which contains **Q** as a subroutine
- showing that if **Q** gives the answer "**yes**", we reach a contradiction
- so **Q** must give the answer "**no**", but this also leads to a contradiction
- the contradiction stems from assuming that **Q**, and hence A, exists
- therefore no algorithm A exists and HP is undecidable

Notice we are not concerned with the complexity of A, just the existence of A