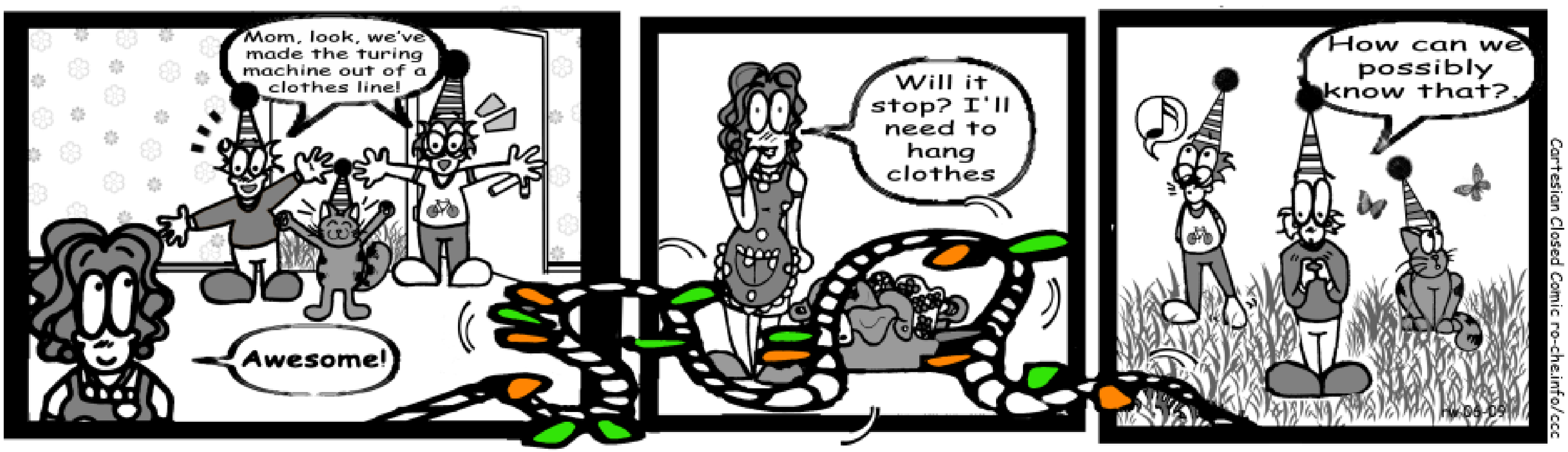
Foundations of Computer Science
Lecture 27

Unsolvable Problems

No Automatic Program Verifier for Hello-World
No Ultimate Debugger or Algorithm for PCP
The Complexity Zoo

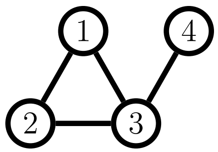


Last Time: Turing Machines

$$\begin{array}{rcl} \text{Intuitive notion of algorithm} & \equiv & \text{Turing Machine} \\ \text{Solvable problem} & \equiv & \text{Turing-}\textit{decidable} \end{array}$$

$$\mathcal{L} = \{\langle G\rangle \mid G \text{ is connected}\}$$

$$\langle G\rangle = \texttt{2; 1; 3; 4 \# 1,2; 2,3; 1,3; 3,4}$$



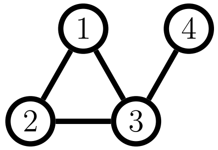
($\langle G\rangle$ is the encoding of graph G as a string.)

Last Time: Turing Machines

Intuitive notion of algorithm	$\equiv$	Turing Machine
Solvable problem	$\equiv$	Turing-*decidable*

$$\mathcal{L} = \{\langle G\rangle \mid G \text{ is connected}\}$$

$\langle G\rangle =$ `2; 1; 3; 4 # 1,2; 2,3; 1,3; 3,4`



($\langle G\rangle$ is the encoding of graph G as a string.)

M = Turing Machine that solves graph connectivity

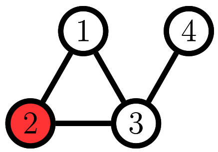
input: $\langle G\rangle$, the encoding of a graph G.

Last Time: Turing Machines

Intuitive notion of algorithm	$\equiv$	Turing Machine
Solvable problem	$\equiv$	Turing-*decidable*

$$\mathcal{L} = \{\langle G \rangle \mid G \text{ is connected}\}$$

$$\langle G \rangle = \texttt{2; 1; 3; 4 \# 1,2; 2,3; 1,3; 3,4}$$



($\langle G \rangle$ is the encoding of graph G as a string.)

M = Turing Machine that solves graph connectivity

input: $\langle G \rangle$, the encoding of a graph G.

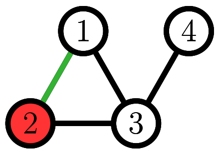
1: Check that $\langle G \rangle$ is a valid encoding of a graph and mark the first vertex in G.

Last Time: Turing Machines

Intuitive notion of algorithm	$\equiv$	Turing Machine
Solvable problem	$\equiv$	Turing-*decidable*

$$\mathcal{L} = \{\langle G\rangle \mid G \text{ is connected}\}$$

$$\langle G\rangle = \texttt{2; 1; 3; 4 \# 1,2; 2,3; 1,3; 3,4}$$



($\langle G\rangle$ is the encoding of graph G as a string.)

M = Turing Machine that solves graph connectivity

input: $\langle G\rangle$, the encoding of a graph G.

1: Check that $\langle G\rangle$ is a valid encoding of a graph and mark the first vertex in G.

2: REPEAT: Find an edge in G between a marked and an unmarked vertex.
Mark the unmarked node or GOTO step 3 if there is no such edge.

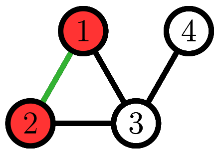
Last Time: Turing Machines

Intuitive notion of algorithm	$\equiv$	Turing Machine
Solvable problem	$\equiv$	Turing-*decidable*

$$\mathcal{L} = \{\langle G\rangle \mid G \text{ is connected}\}$$

$$\langle G\rangle = \texttt{2; 1; 3; 4 \# 1,2; 2,3; 1,3; 3,4}$$

($\langle G\rangle$ is the encoding of graph G as a string.)



M = Turing Machine that solves graph connectivity

input: $\langle G\rangle$, the encoding of a graph G.

1: Check that $\langle G\rangle$ is a valid encoding of a graph and mark the first vertex in G.

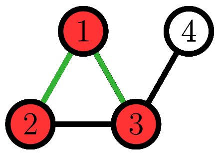
2: REPEAT: Find an edge in G between a marked and an unmarked vertex.
Mark the unmarked node or GOTO step 3 if there is no such edge.

Last Time: Turing Machines

Intuitive notion of algorithm	$\equiv$	Turing Machine
Solvable problem	$\equiv$	Turing-*decidable*

$$\mathcal{L} = \{\langle G\rangle \mid G \text{ is connected}\}$$

$$\langle G\rangle = \texttt{2; 1; 3; 4 \# 1,2; 2,3; 1,3; 3,4}$$



($\langle G\rangle$ is the encoding of graph G as a string.)

M = Turing Machine that solves graph connectivity

input: $\langle G\rangle$, the encoding of a graph G.

1: Check that $\langle G\rangle$ is a valid encoding of a graph and mark the first vertex in G.

2: REPEAT: Find an edge in G between a marked and an unmarked vertex.
Mark the unmarked node or GOTO step 3 if there is no such edge.

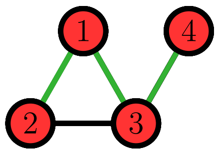
Last Time: Turing Machines

Intuitive notion of algorithm	$\equiv$	Turing Machine
Solvable problem	$\equiv$	Turing-*decidable*

$$\mathcal{L} = \{\langle G\rangle \mid G \text{ is connected}\}$$

$$\langle G\rangle = \texttt{2; 1; 3; 4 \# 1,2; 2,3; 1,3; 3,4}$$

($\langle G\rangle$ is the encoding of graph G as a string.)



M = Turing Machine that solves graph connectivity

input: $\langle G\rangle$, the encoding of a graph G.

1: Check that $\langle G\rangle$ is a valid encoding of a graph and mark the first vertex in G.

2: REPEAT: Find an edge in G between a marked and an unmarked vertex.
Mark the unmarked node or GOTO step 3 if there is no such edge.

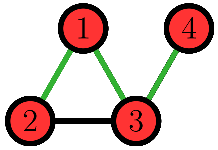
Last Time: Turing Machines

Intuitive notion of algorithm	$\equiv$	Turing Machine
Solvable problem	$\equiv$	Turing-*decidable*

$$\mathcal{L} = \{\langle G \rangle \mid G \text{ is connected}\}$$

$$\langle G \rangle = \texttt{2; 1; 3; 4 \# 1,2; 2,3; 1,3; 3,4}$$

($\langle G \rangle$ is the encoding of graph G as a string.)



M = Turing Machine that solves graph connectivity

input: $\langle G \rangle$, the encoding of a graph G.

1: Check that $\langle G \rangle$ is a valid encoding of a graph and mark the first vertex in G.
2: REPEAT: Find an edge in G between a marked and an unmarked vertex.
Mark the unmarked node or GOTO step 3 if there is no such edge.
3: REJECT if there is an unmarked vertex remaining in G; otherwise ACCEPT.

To tell your friend on the other coast about this fancy Turing Machine M, encode its description into the bit-string $\langle M \rangle$ and send over the telegraph.

You want to solve a different problem? Build another Turing Machine!

Today: Unsolvable Problems

1. Programmable Turing Machines.
2. Examples of unsolvable problems.
   - Post's Correspondence Problem (PCP)?
   - HalfSum?
   - Auto-Grade?
   - Ultimate-Debugger?
3. $\mathcal{L}_{\text{TM}}$: The language recognized by a Universal Turing Machine.
   - $\mathcal{L}_{\text{TM}}$ is undecidable – cannot be solved!
4. Auto-Grade and Ultimate-Debugger do not exist.
5. What about HalfSum?

Programmable Turing Machine: Universal Turing Machine

A Turing Machine M has a binary encoding $\langle M \rangle$. Its input w is a binary string.

Programmable Turing Machine: Universal Turing Machine

A Turing Machine M has a binary encoding $\langle M\rangle$. Its input w is a binary string.

$\langle M\rangle\texttt{\#}w$ can be the input to another Turing Machine U_{TM}.

$$U_{\text{TM}}(\langle M\rangle\texttt{\#}w) \;=\; \begin{cases}\text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT};\\ \text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT};\\ \text{loop forever} & \text{if } M(w) = \text{loop forever};\end{cases}$$

U_{TM} outputs on $\langle M\rangle\texttt{\#}w$ whatever M outputs on w. U_{TM} *simulates* M

Programmable Turing Machine: Universal Turing Machine

A Turing Machine M has a binary encoding $\langle M\rangle$. Its input w is a binary string.

$\langle M\rangle\texttt{\#}w$ can be the input to another Turing Machine U_{TM}.

$$\underbrace{U_{\text{TM}}}_{\text{computer}}(\langle M\rangle\texttt{\#}w) \;=\; \begin{cases} \text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT;} \\ \text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT;} \\ \text{loop forever} & \text{if } M(w) = \text{loop forever;} \end{cases}$$

$\boxed{U_{\text{TM}} \text{ outputs on } \langle M\rangle\texttt{\#}w \text{ whatever } M \text{ outputs on } w.\ U_{\text{TM}}\ \textit{simulates}\ M}$

Programmable Turing Machine: Universal Turing Machine

A Turing Machine M has a binary encoding $\langle M\rangle$. Its input w is a binary string.

$\langle M\rangle\texttt{#}w$ can be the input to another Turing Machine U_{TM}.

$$\underbrace{U_{\text{TM}}}_{\text{computer}}(\underbrace{\langle M\rangle}_{\text{program}}\texttt{#}w) \;=\; \begin{cases} \text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT;} \\ \text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT;} \\ \text{loop forever} & \text{if } M(w) = \text{loop forever;} \end{cases}$$

U_{TM} outputs on $\langle M\rangle\texttt{#}w$ whatever M outputs on w. U_{TM} *simulates* M

Programmable Turing Machine: Universal Turing Machine

A Turing Machine M has a binary encoding $\langle M \rangle$. Its input w is a binary string.

$\langle M \rangle \#w$ can be the input to another Turing Machine U_{TM}.

$$\underbrace{U_{\text{TM}}}_{\text{computer}}(\underbrace{\langle M \rangle}_{\text{program}}\#\underbrace{w}_{\text{program input}}) = \begin{cases} \text{halt with } \textsc{accept} & \text{if } M(w) = \text{halt with } \textsc{accept}; \\ \text{halt with } \textsc{reject} & \text{if } M(w) = \text{halt with } \textsc{reject}; \\ \text{loop forever} & \text{if } M(w) = \text{loop forever}; \end{cases}$$

U_{TM} outputs on $\langle M \rangle \#w$ whatever M outputs on w. U_{TM} *simulates* M

Programmable Turing Machine: Universal Turing Machine

A Turing Machine M has a binary encoding $\langle M \rangle$. Its input w is a binary string.

$\langle M \rangle \#w$ can be the input to another Turing Machine U_{TM}.

$$\underbrace{U_{\text{TM}}}_{\text{computer}}(\underbrace{\langle M \rangle}_{\text{program}}\#\underbrace{w}_{\text{program input}}) = \begin{cases} \text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT}; \\ \text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT}; \\ \text{loop forever} & \text{if } M(w) = \text{loop forever}; \end{cases}$$

U_{TM} outputs on $\langle M \rangle \#w$ whatever M outputs on w. U_{TM} *simulates* M

Challenge: U_{TM} is fixed but can simulate any M, even one with a million states.

Programmable Turing Machine: Universal Turing Machine

A Turing Machine M has a binary encoding $\langle M\rangle$. Its input w is a binary string.

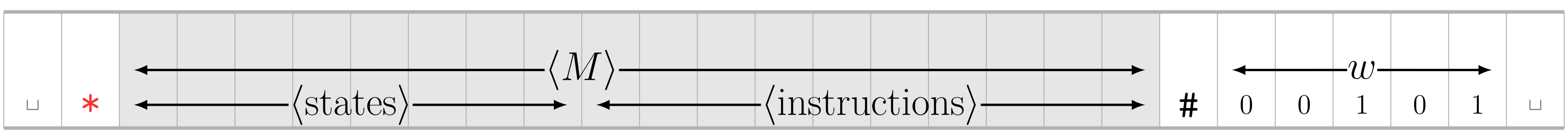
$\langle M\rangle\#w$ can be the input to another Turing Machine U_{TM}.

$$U_{\text{TM}}(\langle M\rangle\#w) = \begin{cases} \text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT;} \\ \text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT;} \\ \text{loop forever} & \text{if } M(w) = \text{loop forever;} \end{cases}$$

computer program program input

U_{TM} outputs on $\langle M\rangle\#w$ whatever M outputs on w. U_{TM} *simulates* M

Challenge: U_{TM} is fixed but can simulate any M, even one with a million states.



Entire simulation is done on the tape.

Programmable Turing Machine: Universal Turing Machine

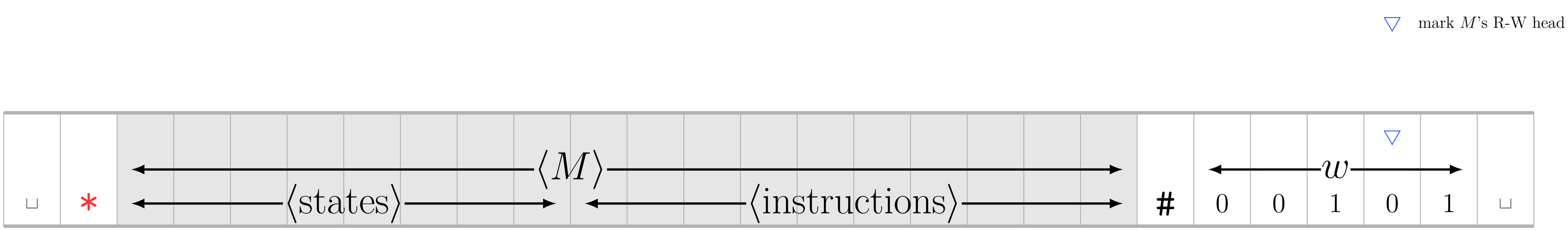
A Turing Machine M has a binary encoding $\langle M\rangle$. Its input w is a binary string.

$\langle M\rangle\#w$ can be the input to another Turing Machine U_{TM}.

$$\underset{\text{computer}}{U_{\text{TM}}}(\underset{\text{program}}{\langle M\rangle}\#\underset{\text{program input}}{w}) = \begin{cases} \text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT;} \\ \text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT;} \\ \text{loop forever} & \text{if } M(w) = \text{loop forever;} \end{cases}$$

U_{TM} outputs on $\langle M\rangle\#w$ whatever M outputs on w. U_{TM} *simulates* M

Challenge: U_{TM} is fixed but can simulate any M, even one with a million states.



Entire simulation is done on the tape.

Programmable Turing Machine: Universal Turing Machine

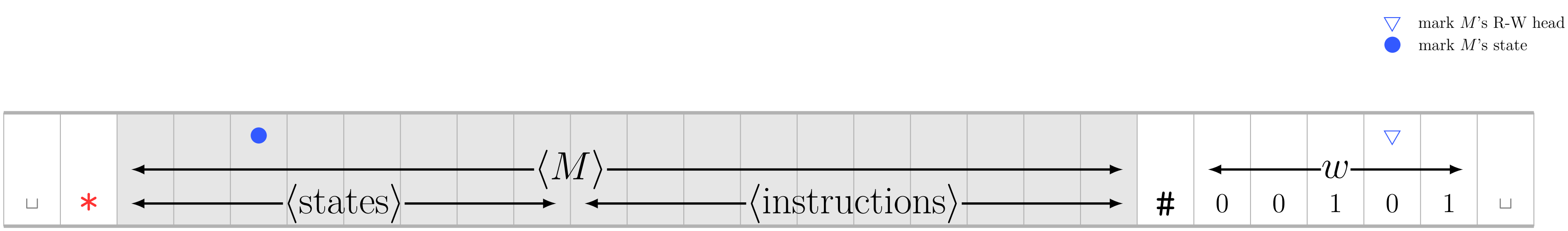
A Turing Machine M has a binary encoding $\langle M\rangle$. Its input w is a binary string.

$\langle M\rangle\#w$ can be the input to another Turing Machine U_{TM}.

$$\underbrace{U_{\text{TM}}}_{\text{computer}}(\underbrace{\langle M\rangle}_{\text{program}}\#\underbrace{w}_{\text{program input}}) = \begin{cases} \text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT;} \\ \text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT;} \\ \text{loop forever} & \text{if } M(w) = \text{loop forever;} \end{cases}$$

U_{TM} outputs on $\langle M\rangle\#w$ whatever M outputs on w. U_{TM} *simulates* M

Challenge: U_{TM} is fixed but can simulate any M, even one with a million states.



Entire simulation is done on the tape.

Programmable Turing Machine: Universal Turing Machine

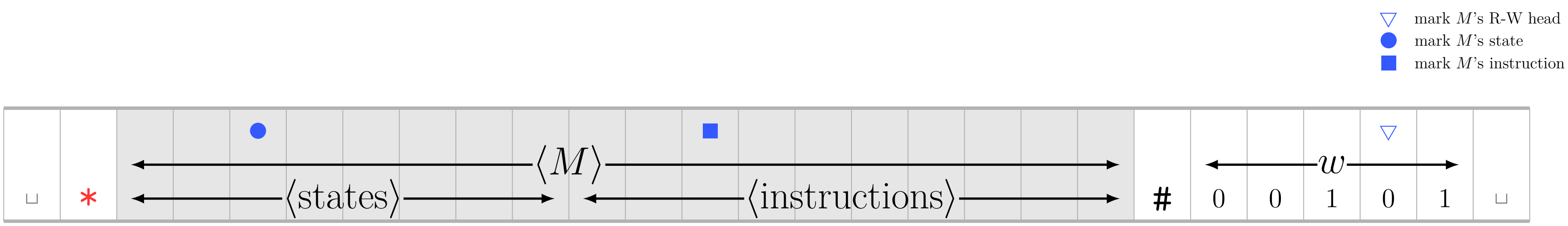
A Turing Machine M has a binary encoding $\langle M\rangle$. Its input w is a binary string.

$\langle M\rangle\#w$ can be the input to another Turing Machine U_{TM}.

$$\underbrace{U_{\text{TM}}}_{\text{computer}}(\underbrace{\langle M\rangle}_{\text{program}}\#\underbrace{w}_{\text{program input}}) = \begin{cases} \text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT;} \\ \text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT;} \\ \text{loop forever} & \text{if } M(w) = \text{loop forever;} \end{cases}$$

U_{TM} outputs on $\langle M\rangle\#w$ whatever M outputs on w. U_{TM} *simulates* M

Challenge: U_{TM} is fixed but can simulate any M, even one with a million states.



Entire simulation is done on the tape.

Programmable Turing Machine: Universal Turing Machine

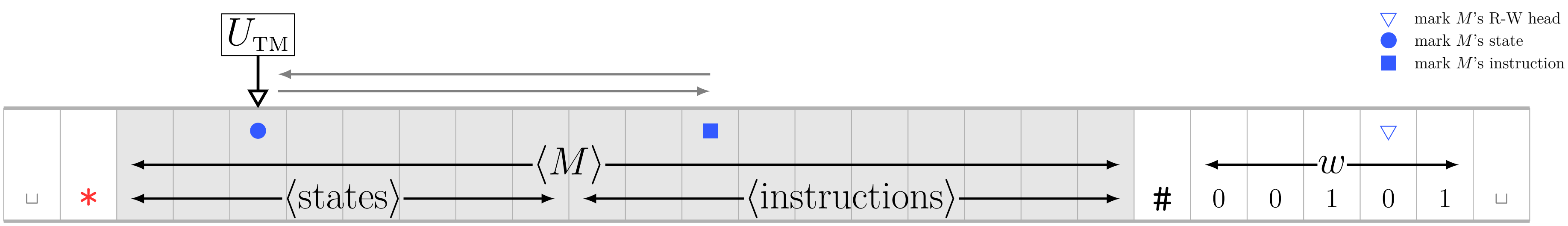
A Turing Machine M has a binary encoding $\langle M \rangle$. Its input w is a binary string.

$\langle M \rangle \texttt{\#} w$ can be the input to another Turing Machine U_{TM}.

$$\underbrace{U_{\text{TM}}}_{\text{computer}}(\underbrace{\langle M \rangle}_{\text{program}}\texttt{\#}\underbrace{w}_{\text{program input}}) = \begin{cases} \text{halt with } \textsc{accept} & \text{if } M(w) = \text{halt with } \textsc{accept}; \\ \text{halt with } \textsc{reject} & \text{if } M(w) = \text{halt with } \textsc{reject}; \\ \text{loop forever} & \text{if } M(w) = \text{loop forever}; \end{cases}$$

U_{TM} outputs on $\langle M \rangle \texttt{\#} w$ whatever M outputs on w. U_{TM} *simulates* M

Challenge: U_{TM} is fixed but can simulate any M, even one with a million states.



Entire simulation is done on the tape.

Post's Correspondence Problem (PCP) and HalfSum

PCP: Consider 3 dominos:

d_1	d_2	d_3
0	01	110
100	00	11

Post's Correspondence Problem (PCP) and HalfSum

PCP: Consider 3 dominos:
$$\begin{array}{ccc} d_1 & d_2 & d_3 \\ \boxed{\begin{array}{c} 0 \\ \hline 100 \end{array}} & \boxed{\begin{array}{c} 01 \\ \hline 00 \end{array}} & \boxed{\begin{array}{c} 110 \\ \hline 11 \end{array}} \end{array}$$

$$d_3 d_2 d_3 d_1 \;=\; \boxed{\begin{array}{c} 110 \\ \hline 11 \end{array}}\boxed{\begin{array}{c} 01 \\ \hline 00 \end{array}}\boxed{\begin{array}{c} 110 \\ \hline 11 \end{array}}\boxed{\begin{array}{c} 0 \\ \hline 100 \end{array}} \;=\; \boxed{\begin{array}{c} 110011100 \\ \hline 110011100 \end{array}}$$

$\leftarrow$ Top and bottom strings match. That's the goal.

Post's Correspondence Problem (PCP) and HalfSum

PCP: Consider 3 dominos: $\underset{d_1}{\left[\frac{0}{100}\right]}\ \underset{d_2}{\left[\frac{01}{00}\right]}\ \underset{d_3}{\left[\frac{110}{11}\right]}$

$$d_3d_2d_3d_1 = \left[\frac{110}{11}\right]\left[\frac{01}{00}\right]\left[\frac{110}{11}\right]\left[\frac{0}{100}\right] = \left[\frac{110011100}{110011100}\right]$$

$\leftarrow$ Top and bottom strings match. That's the goal.

INPUT: Dominos $\{d_1, d_2, \ldots, d_n\}$. For example $\left\{\left[\frac{10}{101}\right], \left[\frac{011}{11}\right], \left[\frac{101}{011}\right]\right\}$.

TASK: Can one line up finitely many dominos so that the top and bottom strings match?

Post's Correspondence Problem (PCP) and HalfSum

PCP: Consider 3 dominos: $\overset{d_1}{\begin{array}{|c|}\hline 0\\ \hline 100\\ \hline\end{array}}\ \overset{d_2}{\begin{array}{|c|}\hline 01\\ \hline 00\\ \hline\end{array}}\ \overset{d_3}{\begin{array}{|c|}\hline 110\\ \hline 11\\ \hline\end{array}}$

$$d_3d_2d_3d_1 = \begin{array}{|c|c|c|c|}\hline 110 & 01 & 110 & 0\\ \hline 11 & 00 & 11 & 100\\ \hline\end{array} = \begin{array}{|c|}\hline 110011100\\ \hline 110011100\\ \hline\end{array}$$

← Top and bottom strings match. That's the goal.

INPUT: Dominos $\{d_1, d_2, \ldots, d_n\}$. For example $\left\{\begin{array}{|c|}\hline 10\\ \hline 101\\ \hline\end{array}, \begin{array}{|c|}\hline 011\\ \hline 11\\ \hline\end{array}, \begin{array}{|c|}\hline 101\\ \hline 011\\ \hline\end{array}\right\}$.

TASK: Can one line up finitely many dominos so that the top and bottom strings match?

HalfSum: Consider the multiset $S = \{1, 1, 1, 3, 4, 4, 5, 6, 9\}$, and subset $A = \{1, 3, 4, 9\}$.

$$\text{sum}(A) = 17 = \tfrac{1}{2} \times \text{sum}(S).$$

Post's Correspondence Problem (PCP) and HALFSUM

PCP: Consider 3 dominos: $\underset{d_1}{\left[\frac{0}{100}\right]}\ \underset{d_2}{\left[\frac{01}{00}\right]}\ \underset{d_3}{\left[\frac{110}{11}\right]}$

$$d_3d_2d_3d_1 = \left[\frac{110}{11}\right]\left[\frac{01}{00}\right]\left[\frac{110}{11}\right]\left[\frac{0}{100}\right] = \left[\frac{110011100}{110011100}\right]$$

← Top and bottom strings match. That's the goal.

INPUT: Dominos $\{d_1, d_2, \ldots, d_n\}$. For example $\left\{\left[\frac{10}{101}\right], \left[\frac{011}{11}\right], \left[\frac{101}{011}\right]\right\}$.

TASK: Can one line up finitely many dominos so that the top and bottom strings match?

HalfSum: Consider the multiset $S = \{1, 1, 1, 3, 4, 4, 5, 6, 9\}$, and subset $A = \{1, 3, 4, 9\}$.

$$\text{sum}(A) = 17 = \tfrac{1}{2} \times \text{sum}(S).$$

INPUT: Multiset $S = \{x_1, x_2, \ldots, x_n\}$. For example, $S = \{1, 1, 1, 3, 4, 4, 5, 6, 9\}$.

TASK: Is there a subset whose sum is $\frac{1}{2} \times \text{sum}(S) = \frac{1}{2} \times (x_1 + x_2 + \cdots + x_n)$?

Auto-Grade

Your first CS assignment: Write a program to print "Hello World!" and halt.

CS1: 700+ submissions!

Naturally, we do not grade these by hand.

Auto-Grade: runs each submission and determines if its correct. ←**program verification**

Auto-Grade

Your first CS assignment: Write a program to print "Hello World!" and halt.

CS1: 700+ submissions!

Naturally, we do not grade these by hand.

Auto-Grade: runs each submission and determines if its correct. ←**program verification**

What does Auto-Grade say for this program:

```
n = 4;
while(n > 0){
  if(n is not a sum of two primes){
    print("Hello World!") and exit;
  }
n ← n + 2;
}
```

ULTIMATE-DEBUGGER

Wouldn't it be nice to have the ULTIMATE-DEBUGGER. ← solves the *Halting Problem*

HALTS(
```
n = 4;
while(n > 0){
  if(n is not a sum of two primes){
    print("Hello World!") and exit;
  }
n ← n + 2;
}
```
) $= \begin{cases} \boxed{\text{YES}} & \text{if program halts} \\ \boxed{\text{NO}} & \text{if program infinitely loops} \end{cases}$

ULTIMATE-DEBUGGER

Wouldn't it be nice to have the ULTIMATE-DEBUGGER. ← solves the *Halting Problem*

$$\text{HALTS}\left(\boxed{\begin{array}{l}\texttt{n = 4;}\\ \texttt{while(n > 0)\{}\\ \quad\texttt{if(n is not a sum of two primes)\{}\\ \qquad\texttt{print("Hello World!") and exit;}\\ \quad\texttt{\}}\\ \texttt{n} \leftarrow \texttt{n + 2;}\\ \texttt{\}}\end{array}}\right) = \begin{cases}\boxed{\text{YES}} & \text{if program halts}\\ \boxed{\text{NO}} & \text{if program infinitely loops}\end{cases}$$

- We can grade the students program correctly.
- We can solve Goldbach's conjecture.
- Just think what you could do with ULTIMATE-DEBUGGER.
  - No more infinite looping programs.

Verification: Does A Program Successfully Terminate?

Verification: Does A Program Successfully Terminate?

$$\mathcal{L}_{\text{TM}} = \{\langle M\rangle\texttt{\#}w \mid M \text{ is a Turing Machine and } M \text{ accepts } w\}.$$

Verification: Does A Program Successfully Terminate?

$$\mathcal{L}_{\text{TM}} = \{\langle M\rangle\texttt{\#}w \mid M \text{ is a Turing Machine and } M \text{ accepts } w\}.$$

U_{TM} is a *recognizer* for $\mathcal{L}_{\text{TM}}$.

Verification: Does A Program Successfully Terminate?

$$\mathcal{L}_{\text{TM}} = \{\langle M\rangle\texttt{\#}w \mid M \text{ is a Turing Machine and } M \text{ accepts } w\}.$$

U_{TM} is a *recognizer* for $\mathcal{L}_{\text{TM}}$.

Is there a Turing Machine A_{TM} which **decides** $\mathcal{L}_{\text{TM}}$?

- A decider must *always* halt with an answer.
- U_{TM} may loop forever if M loops forever on w.

Verification: Does A Program Successfully Terminate?

$$\mathcal{L}_{\text{TM}} = \{\langle M\rangle\texttt{\#}w \mid M \text{ is a Turing Machine and } M \text{ accepts } w\}.$$

U_{TM} is a *recognizer* for $\mathcal{L}_{\text{TM}}$.

Is there a Turing Machine A_{TM} which **decides** $\mathcal{L}_{\text{TM}}$?

- A decider must *always* halt with an answer.
- U_{TM} may loop forever if M loops forever on w.
- Question: What do these mean: $M(\langle M\rangle)$ and $A_{\text{TM}}(\langle M\rangle\texttt{\#}\langle M\rangle)$?

Verification: Does A Program Successfully Terminate?

$$\mathcal{L}_{\text{TM}} = \{\langle M\rangle\#w \mid M \text{ is a Turing Machine and } M \text{ accepts } w\}.$$

U_{TM} is a *recognizer* for $\mathcal{L}_{\text{TM}}$.

Is there a Turing Machine A_{TM} which **decides** $\mathcal{L}_{\text{TM}}$?

- A decider must *always* halt with an answer.
- U_{TM} may loop forever if M loops forever on w.
- Question: What do these mean: $M(\langle M\rangle)$ and $A_{\text{TM}}(\langle M\rangle\#\langle M\rangle)$?

A diabolical Turing Machine D built from A_{TM}:

D = "Diagonal" Turing Machine derived from A_{TM} (the decider for $\mathcal{L}_{\text{TM}}$)

input: $\langle M\rangle$ where M is a Turing Machine.

Verification: Does A Program Successfully Terminate?

$$\mathcal{L}_{\text{TM}} = \{\langle M \rangle \# w \mid M \text{ is a Turing Machine and } M \text{ accepts } w\}.$$

U_{TM} is a *recognizer* for $\mathcal{L}_{\text{TM}}$.

Is there a Turing Machine A_{TM} which **decides** $\mathcal{L}_{\text{TM}}$?

- A decider must *always* halt with an answer.
- U_{TM} may loop forever if M loops forever on w.
- Question: What do these mean: $M(\langle M \rangle)$ and $A_{\text{TM}}(\langle M \rangle \# \langle M \rangle)$?

A diabolical Turing Machine D built from A_{TM}:

D = "Diagonal" Turing Machine derived from A_{TM} (the decider for $\mathcal{L}_{\text{TM}}$)

input: $\langle M \rangle$ where M is a Turing Machine.

1: Run A_{TM} with input $\langle M \rangle \# \langle M \rangle$.

D does the *opposite* of A_{TM}. Is D a decider?

Verification: Does A Program Successfully Terminate?

$$\mathcal{L}_{\text{TM}} = \{\langle M\rangle\#w \mid M \text{ is a Turing Machine and } M \text{ accepts } w\}.$$

U_{TM} is a *recognizer* for $\mathcal{L}_{\text{TM}}$.

Is there a Turing Machine A_{TM} which **decides** $\mathcal{L}_{\text{TM}}$?

- A decider must *always* halt with an answer.
- U_{TM} may loop forever if M loops forever on w.
- Question: What do these mean: $M(\langle M\rangle)$ and $A_{\text{TM}}(\langle M\rangle\#\langle M\rangle)$?

A diabolical Turing Machine D built from A_{TM}:

D = "Diagonal" Turing Machine derived from A_{TM} (the decider for $\mathcal{L}_{\text{TM}}$)
input: $\langle M\rangle$ where M is a Turing Machine.
1: Run A_{TM} with input $\langle M\rangle\#\langle M\rangle$.
2: If A_{TM} accepts then REJECT; otherwise (A_{TM} rejects) ACCEPT

D does the *opposite* of A_{TM}. Is D a decider?

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\rightarrow$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\rightarrow$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$						
$\langle M_2\rangle$						
$\langle M_3\rangle$						
$\langle M_4\rangle$						
$\langle D\rangle$						
$\vdots$						

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\rightarrow$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	<u>ACCEPT</u>	ACCEPT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_2\rangle$						
$\langle M_3\rangle$						
$\langle M_4\rangle$						
$\langle D\rangle$						
$\vdots$						

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\rightarrow$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	<u>ACCEPT</u>	ACCEPT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_2\rangle$						
$\langle M_3\rangle$						
$\langle M_4\rangle$						
$\langle D\rangle$	REJECT					
$\vdots$						

$D(\langle M_i\rangle)$ does the *opposite* of $A_{\text{TM}}(\langle M_i\rangle\#\langle M_i\rangle)$.

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\rightarrow$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	<u>ACCEPT</u>	ACCEPT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_2\rangle$	REJECT	<u>REJECT</u>	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_3\rangle$						
$\langle M_4\rangle$						
$\langle D\rangle$	REJECT					
$\vdots$						

$$D(\langle M_i\rangle) \text{ does the } \textit{opposite} \text{ of } A_{\text{TM}}(\langle M_i\rangle\#\langle M_i\rangle).$$

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\to$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	<u>ACCEPT</u>	ACCEPT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_2\rangle$	REJECT	<u>REJECT</u>	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_3\rangle$						
$\langle M_4\rangle$						
$\langle D\rangle$	REJECT	ACCEPT				
$\vdots$						

$$D(\langle M_i\rangle) \text{ does the } \textit{opposite} \text{ of } A_{\text{TM}}(\langle M_i\rangle\#\langle M_i\rangle).$$

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\to$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	ACCEPT	ACCEPT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_2\rangle$	REJECT	REJECT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_3\rangle$	ACCEPT	ACCEPT	REJECT	REJECT	ACCEPT	$\cdots$
$\langle M_4\rangle$						
$\langle D\rangle$	REJECT	ACCEPT				
$\vdots$						

$$D(\langle M_i\rangle) \text{ does the } \textit{opposite} \text{ of } A_{\text{TM}}(\langle M_i\rangle\#\langle M_i\rangle).$$

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\rightarrow$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	ACCEPT	ACCEPT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_2\rangle$	REJECT	REJECT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_3\rangle$	ACCEPT	ACCEPT	REJECT	REJECT	ACCEPT	$\cdots$
$\langle M_4\rangle$						
$\langle D\rangle$	REJECT	ACCEPT	ACCEPT			
$\vdots$						

$D(\langle M_i\rangle)$ does the *opposite* of $A_{\text{TM}}(\langle M_i\rangle\#\langle M_i\rangle)$.

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\rightarrow$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	<u>ACCEPT</u>	ACCEPT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_2\rangle$	REJECT	<u>REJECT</u>	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_3\rangle$	ACCEPT	ACCEPT	<u>REJECT</u>	REJECT	ACCEPT	$\cdots$
$\langle M_4\rangle$	ACCEPT	REJECT	REJECT	<u>REJECT</u>	ACCEPT	$\cdots$
$\langle D\rangle$	REJECT	ACCEPT	ACCEPT			
$\vdots$						

$$D(\langle M_i\rangle) \text{ does the } \textit{opposite} \text{ of } A_{\text{TM}}(\langle M_i\rangle\#\langle M_i\rangle).$$

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\rightarrow$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	ACCEPT	ACCEPT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_2\rangle$	REJECT	REJECT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_3\rangle$	ACCEPT	ACCEPT	REJECT	REJECT	ACCEPT	$\cdots$
$\langle M_4\rangle$	ACCEPT	REJECT	REJECT	REJECT	ACCEPT	$\cdots$
$\langle D\rangle$	REJECT	ACCEPT	ACCEPT	ACCEPT		
$\vdots$						

$$D(\langle M_i\rangle) \text{ does the } \textit{opposite} \text{ of } A_{\text{TM}}(\langle M_i\rangle\#\langle M_i\rangle).$$

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\to$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	ACCEPT	ACCEPT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_2\rangle$	REJECT	REJECT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_3\rangle$	ACCEPT	ACCEPT	REJECT	REJECT	ACCEPT	$\cdots$
$\langle M_4\rangle$	ACCEPT	REJECT	REJECT	REJECT	ACCEPT	$\cdots$
$\langle D\rangle$	REJECT	ACCEPT	ACCEPT	ACCEPT	ACCEPT?	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$$D(\langle M_i\rangle) \text{ does the } \textit{opposite} \text{ of } A_{\text{TM}}(\langle M_i\rangle\#\langle M_i\rangle).$$

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\to$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	ACCEPT	ACCEPT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_2\rangle$	REJECT	REJECT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_3\rangle$	ACCEPT	ACCEPT	REJECT	REJECT	ACCEPT	$\cdots$
$\langle M_4\rangle$	ACCEPT	REJECT	REJECT	REJECT	ACCEPT	$\cdots$
$\langle D\rangle$	REJECT	ACCEPT	ACCEPT	ACCEPT	REJECT?	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$D(\langle M_i\rangle)$ does the *opposite* of $A_{\text{TM}}(\langle M_i\rangle\#\langle M_i\rangle)$.

Theorem. A_{TM} does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

A_{TM} exists $\rightarrow$ D exists.

D exists means it will appear on the list of all Turing Machines,

$$\langle M_1\rangle, \langle M_2\rangle, \langle M_3\rangle, \langle M_4\rangle, \langle D\rangle, \ldots$$

Consider what happens when M_i runs on $\langle M_j\rangle$, that is $A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$.

$A_{\text{TM}}(\langle M_i\rangle\#\langle M_j\rangle)$	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle D\rangle$	$\cdots$
$\langle M_1\rangle$	ACCEPT	ACCEPT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_2\rangle$	REJECT	REJECT	REJECT	ACCEPT	ACCEPT	$\cdots$
$\langle M_3\rangle$	ACCEPT	ACCEPT	REJECT	REJECT	ACCEPT	$\cdots$
$\langle M_4\rangle$	ACCEPT	REJECT	REJECT	REJECT	ACCEPT	$\cdots$
$\langle D\rangle$	REJECT	ACCEPT	ACCEPT	ACCEPT	ACCEPT?	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$$D(\langle M_i\rangle) \text{ does the } \textit{opposite} \text{ of } A_{\text{TM}}(\langle M_i\rangle\#\langle M_i\rangle).$$

ULTIMATE-DEBUGGER and AUTO-GRADE Don't Exist

No *general* program/algorithm to analyze *any* other program M and tell if M will accept or not a particular input. ☹

ULTIMATE-DEBUGGER and AUTO-GRADE Don't Exist

No *general* program/algorithm to analyze *any* other program M and tell if M will accept or not a particular input. ☹

No ULTIMATE-DEBUGGER to analyze other programs and tell if they halt. ☹

Suppose ULTIMATE-DEBUGGER H_{TM} exists and *decides* if any other program halts.

We can use H_{TM} to construct a solver A_{TM} for $\mathcal{L}_{\text{TM}}$.

ULTIMATE-DEBUGGER and AUTO-GRADE Don't Exist

No *general* program/algorithm to analyze *any* other program M and tell if M will accept or not a particular input. ☹

No ULTIMATE-DEBUGGER to analyze other programs and tell if they halt. ☹

Suppose ULTIMATE-DEBUGGER H_{TM} exists and *decides* if any other program halts.

We can use H_{TM} to construct a solver A_{TM} for $\mathcal{L}_{\text{TM}}$.

A_{TM} = Turing Machine derived from H_{TM} (the decider for $\mathcal{L}_{\text{HALT}}$)

input: $\langle M\rangle\#w$ where M is a Turing Machine and w an input to M.

ULTIMATE-DEBUGGER and AUTO-GRADE Don't Exist

No *general* program/algorithm to analyze *any* other program M and tell if M will accept or not a particular input. ☹

No ULTIMATE-DEBUGGER to analyze other programs and tell if they halt. ☹

Suppose ULTIMATE-DEBUGGER H_{TM} exists and *decides* if any other program halts.

We can use H_{TM} to construct a solver A_{TM} for $\mathcal{L}_{\text{TM}}$.

A_{TM} = Turing Machine derived from H_{TM} (the decider for $\mathcal{L}_{\text{HALT}}$)

input: $\langle M\rangle\#w$ where M is a Turing Machine and w an input to M.

1: Run H_{TM} on input $\langle M\rangle\#w$. If H_{TM} rejects, then REJECT.

2: Run U_{TM} on input $\langle M\rangle\#w$ and output the decision U_{TM} gives.

ULTIMATE-DEBUGGER and AUTO-GRADE Don't Exist

No *general* program/algorithm to analyze *any* other program M and tell if M will accept or not a particular input. ☹

No ULTIMATE-DEBUGGER to analyze other programs and tell if they halt. ☹

No AUTO-GRADE for CS-1 programs. ☹

Suppose ULTIMATE-DEBUGGER H_{TM} exists and *decides* if any other program halts.

We can use H_{TM} to construct a solver A_{TM} for $\mathcal{L}_{\text{TM}}$.

A_{TM} = Turing Machine derived from H_{TM} (the decider for $\mathcal{L}_{\text{HALT}}$)

input: $\langle M\rangle\#w$ where M is a Turing Machine and w an input to M.

1: Run H_{TM} on input $\langle M\rangle\#w$. If H_{TM} rejects, then REJECT.

2: Run U_{TM} on input $\langle M\rangle\#w$ and output the decision U_{TM} gives.

Exercise. Show that AUTO-GRADE does not exist.

Exercise. Show that HALFSUM is solvable by giving a decider.

Ultimate-Debugger and Auto-Grade Don't Exist

No *general* program/algorithm to analyze *any* other program M and tell if M will accept or not a particular input.

- No Ultimate-Debugger to analyze other programs and tell if they halt.
- No Auto-Grade for CS-1 programs.
- No solver for PCP.

Suppose Ultimate-Debugger H_{TM} exists and *decides* if any other program halts.

We can use H_{TM} to construct a solver A_{TM} for $\mathcal{L}_{\text{TM}}$.

A_{TM} = Turing Machine derived from H_{TM} (the decider for $\mathcal{L}_{\text{HALT}}$)

input: $\langle M\rangle\#w$ where M is a Turing Machine and w an input to M.

1: Run H_{TM} on input $\langle M\rangle\#w$. If H_{TM} rejects, then REJECT.

2: Run U_{TM} on input $\langle M\rangle\#w$ and output the decision U_{TM} gives.

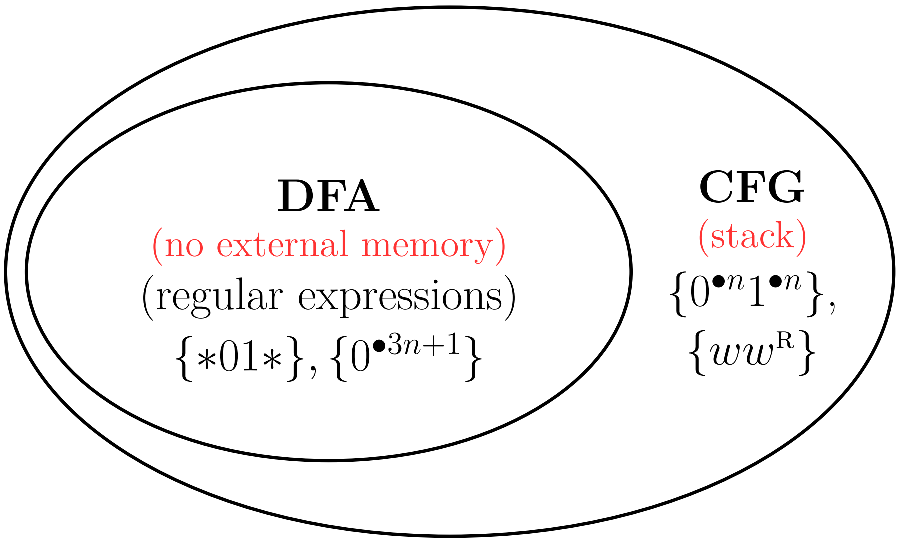
Exercise. Show that Auto-Grade does not exist.

Exercise. Show that HalfSum is solvable by giving a decider.

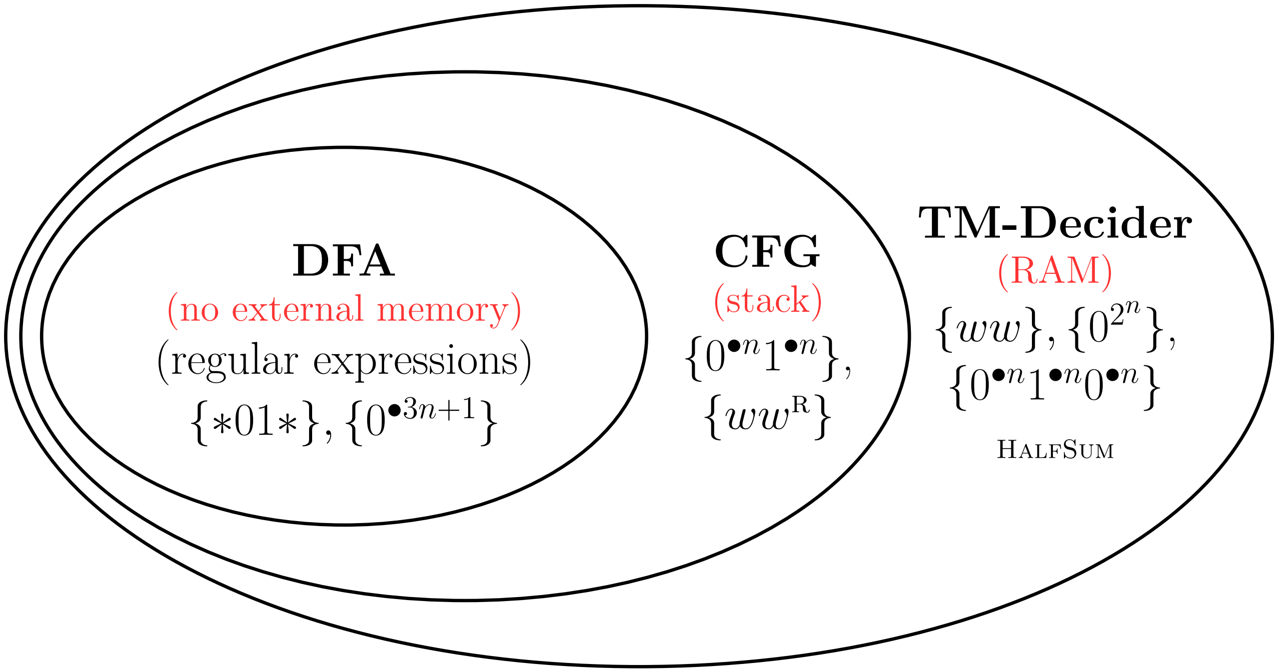
The Landscape

DFA
(no external memory)
(regular expressions)
$\{*01*\}, \{0^{\bullet 3n+1}\}$

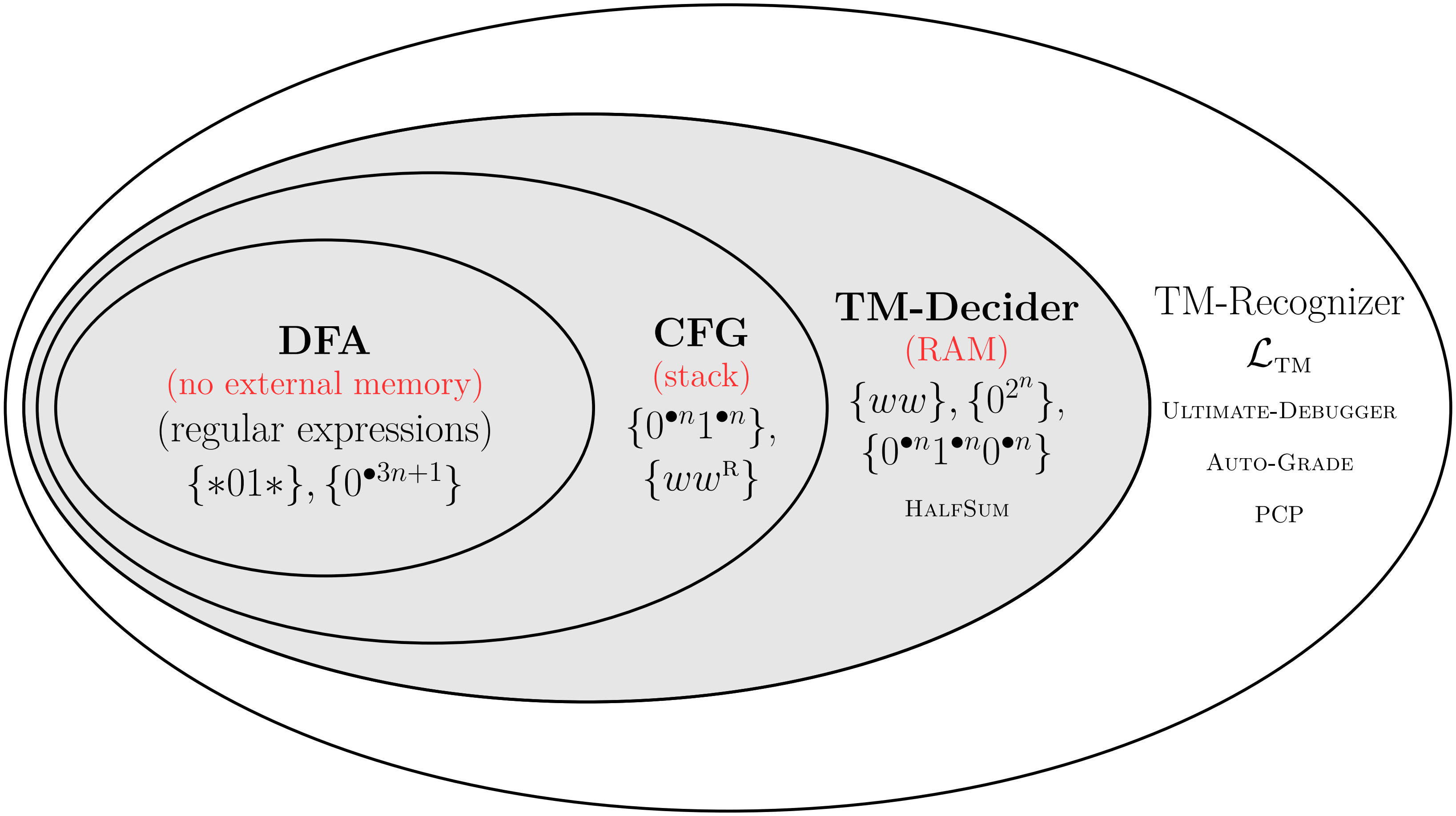
The Landscape



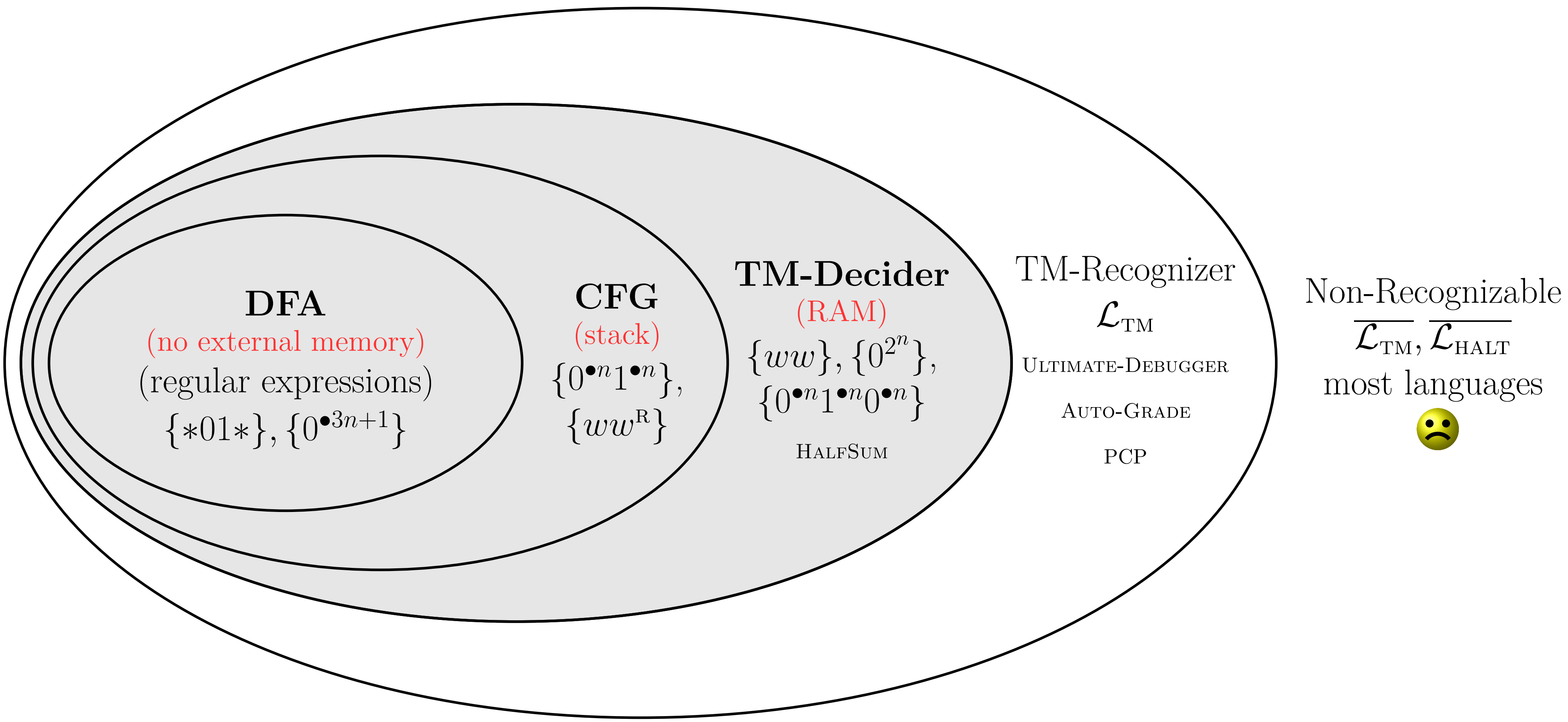
The Landscape



The Landscape



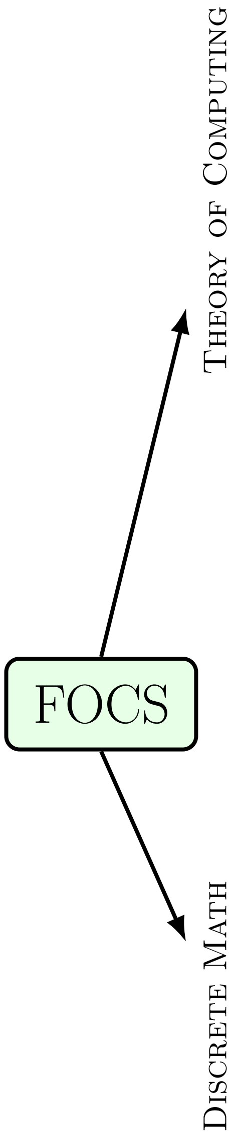
The Landscape



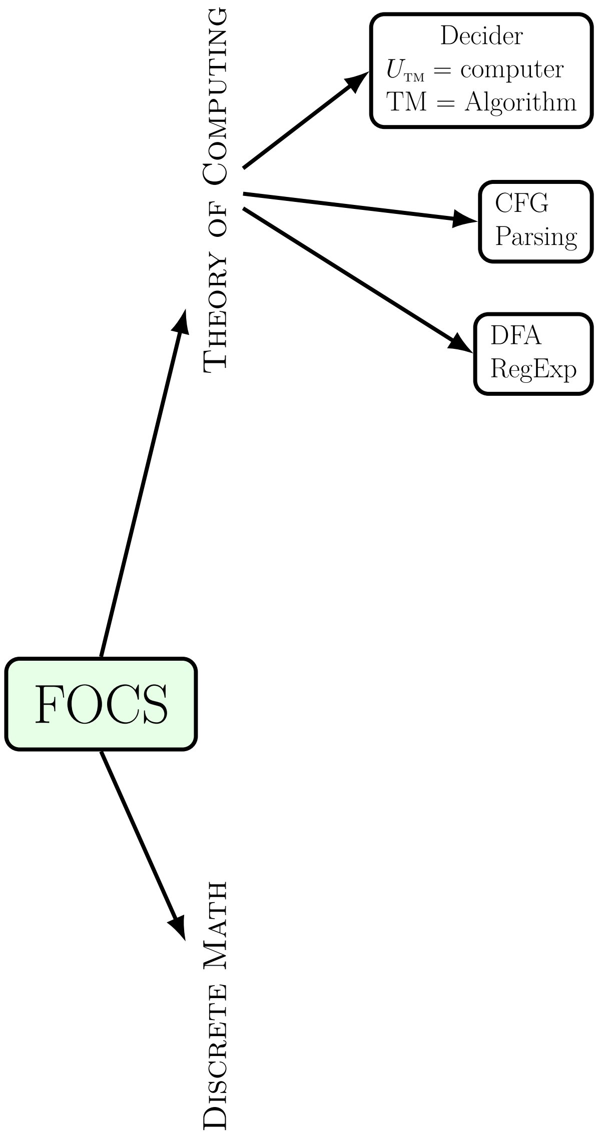
The Path Forward: Focus on Decidable Problems

FOCS

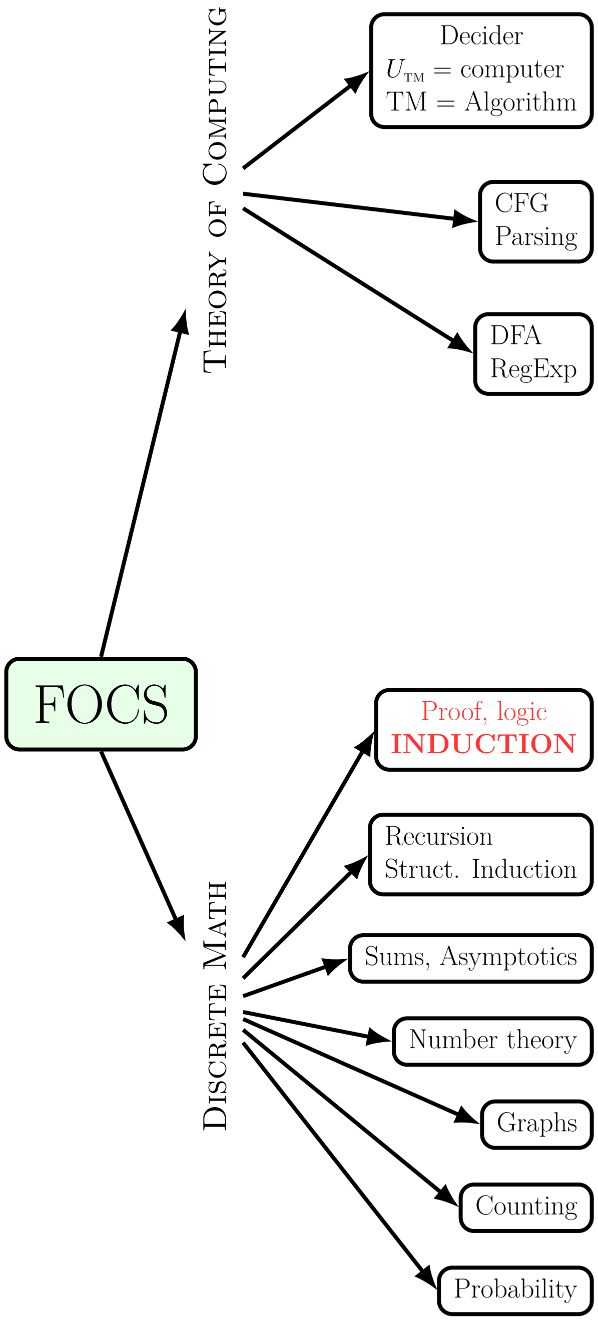
The Path Forward: Focus on Decidable Problems



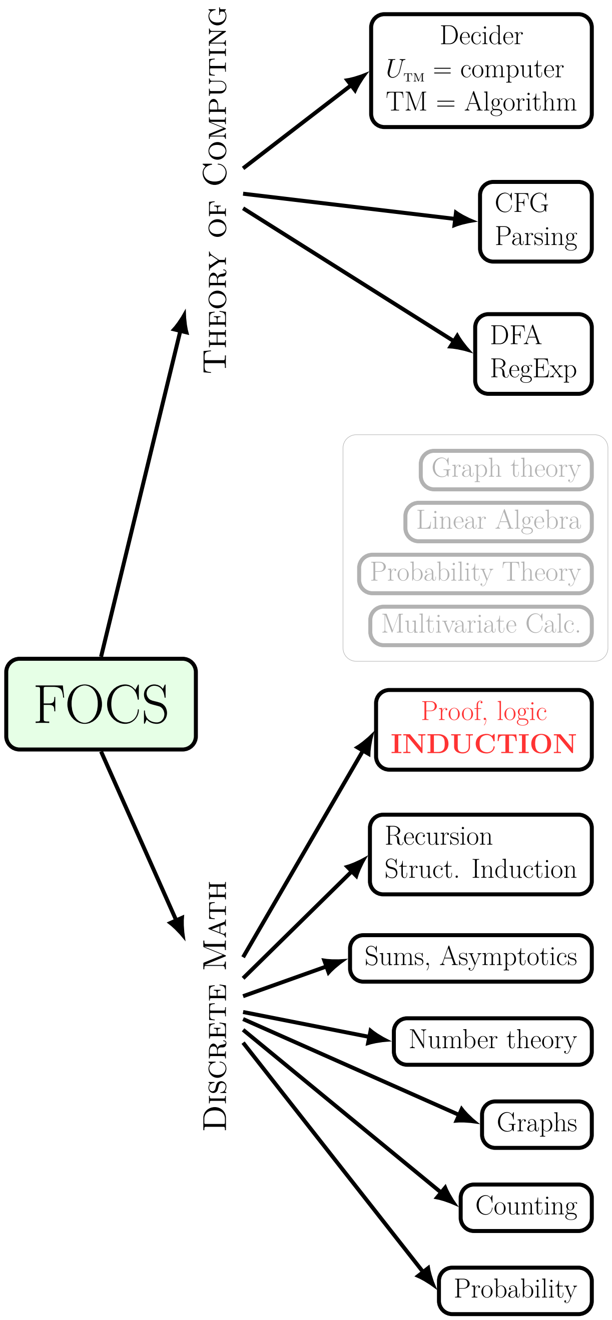
The Path Forward: Focus on Decidable Problems



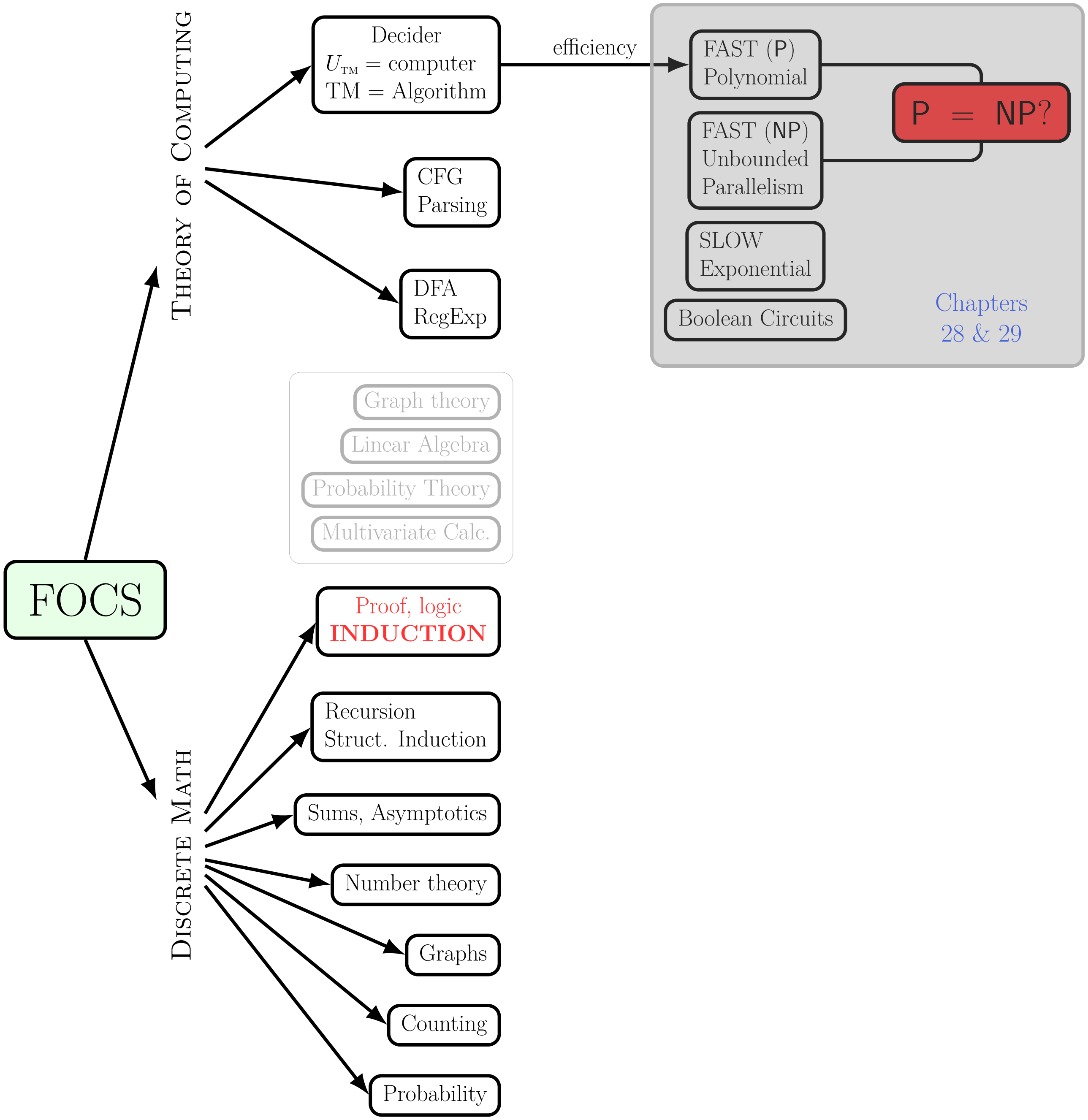
The Path Forward: Focus on Decidable Problems



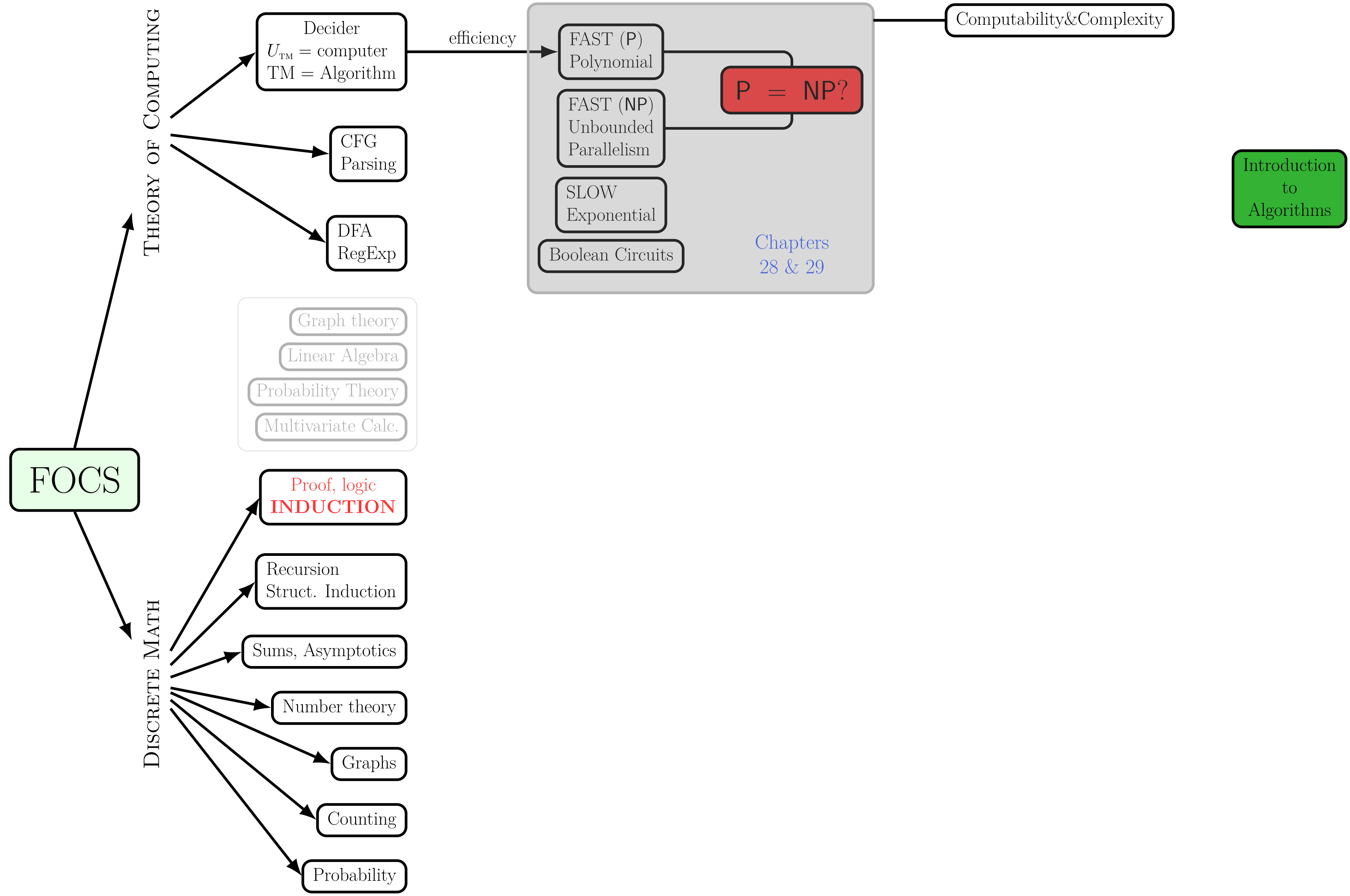
The Path Forward: Focus on Decidable Problems



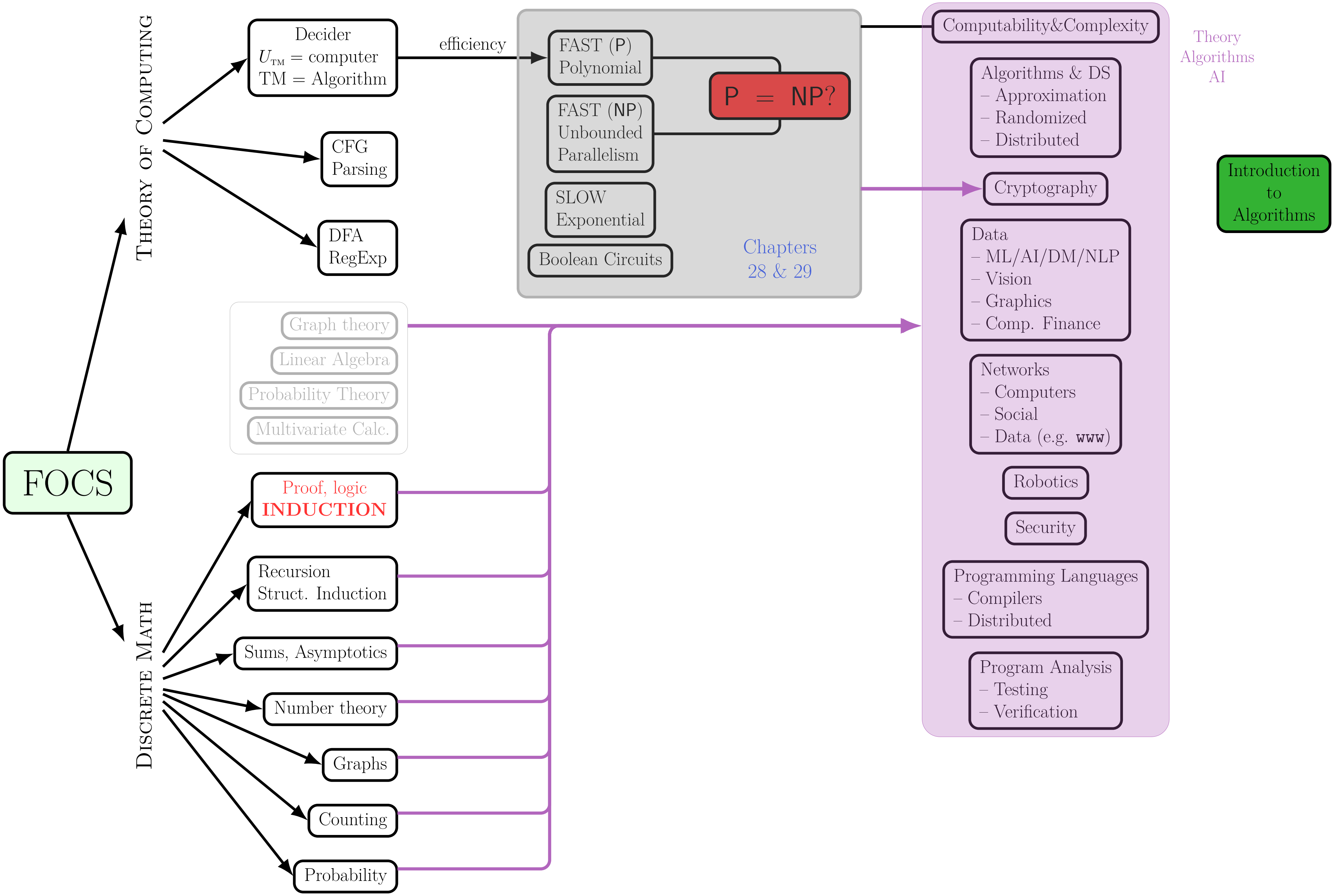
The Path Forward: Focus on Decidable Problems



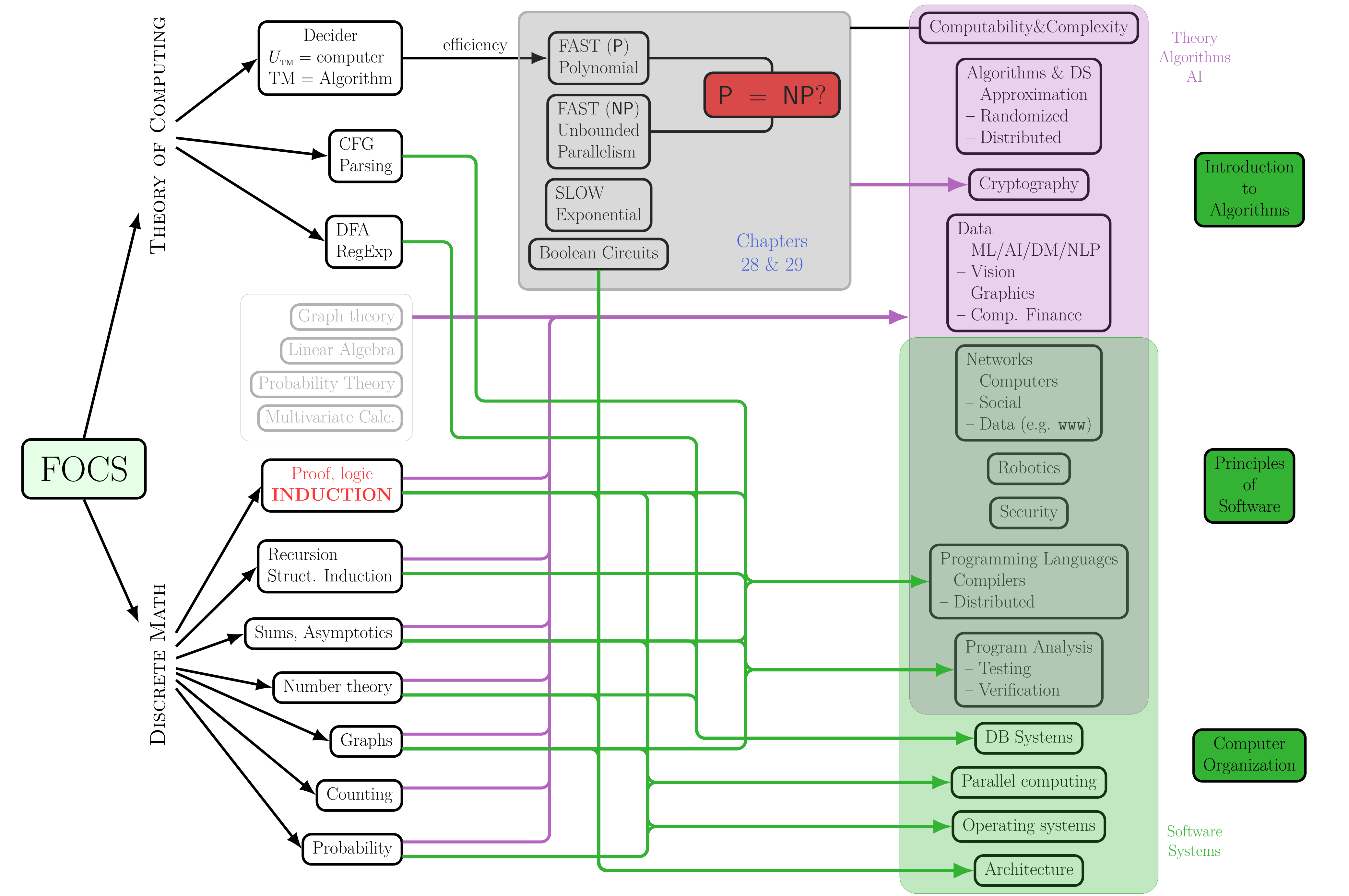
The Path Forward: Focus on Decidable Problems



The Path Forward: Focus on Decidable Problems



The Path Forward: Focus on Decidable Problems



. . . the high technology so celebrated today is essentially a mathematical technology.

"To err is human, but to really foul things up you need a computer." – Paul Ehrlich

. . . the high technology so celebrated today is essentially a mathematical technology.

"To err is human, but to really foul things up you need a computer." – Paul Ehrlich

- **Mariner rocket explodes (1962).** Formula into code bug resulted in no smoothing of deviations.
- **WWWIII (1983)?** Soviet EWS detects 5 US-missiles (bug detected sunlight reflections).
  - **Luckily** *Stanislav "funny feeling in my gut" Petrov* thought: "surely they'd use more missiles?"
- **Therac 25 (1985).** Concurrent programming bug killed patients through massive 100$\times$ radiation overdose.
- **AT&T Lines Go Dead (1990).** 75 million calls dropped (one line of buggy code in software upgrade).
- **Patriot missile defense fails (1991). 28 soldiers dead, 100 injured** (rounding error in scud-detection).
- **Pentium floating point long-division bug (1993).** Cost: $475 million – flawed division table.
- **Ariane rocket explosion (1996).** Cost: $500 million – overflow in 64-bit to 16-bit conversion.
- **Y2K (1999).** Cost: $500 billion spent because year was stored as 2 digits to save space.
- **Mars Climate Orbiter Crash (1998).** Cost: $125 million lost due to metric to imperial units bug.
- **Tesla Self-Driving Car (2016). 1 dead**. Auto-pilot didn't "see" tractor-trailer.
- **Financial Disasters:** London Stock Exchange down due to single server bug (**2009**; billions of pounds of trading); Knight Capital computer glitch trigers stock sale (**2012**; 500 million lost and Knight's value drops by 75%).
- **Airline Disasters:**
  - AirFrance 447 2009, **228 dead**: pitot-tube failure feeds inconsistent data to programs which then panic pilot.
  - Spanair 5022, 2008, **154 dead**: malware virus.
  - AdamAir 574, 2007, **102 dead**: navigation system errors (and pilot errors).
  - KoreanAir 801, 1997, **228 dead**: ground proximity warning system bug.
  - AeroPerú 603, 1996, **70 dead**: altimeter failures.
  - Scottish RAF Chinook, 1994, **29 dead**: faulty test program
  - AirFrance 296, 1988, **3 dead**: altimeter bug.
  - IranAir 655, 1988, **290 dead**: shot down by US Aegis combat system (misidentified as attacking military plane).
  - KoreanAir 007, 1983, **269 dead**: autopilot took plane into Soviet airspace where it got shot down.
  - Boeing 737 Max, 2018,2019, **346 dead**: attack sensor + algorithm errors.

. . . the high technology so celebrated today is essentially a mathematical technology.

"To err is human, but to really foul things up you need a computer." – Paul Ehrlich

- **Mariner rocket explodes (1962).** Formula into code bug resulted in no smoothing of deviations.
- **WWWIII (1983)?** Soviet EWS detects 5 US-missiles (bug detected sunlight reflections).
  - **Luckily** *Stanislav "funny feeling in my gut" Petrov* thought: "surely they'd use more missiles?"
- **Therac 25 (1985).** Concurrent programming bug killed patients through massive 100× radiation overdose.
- **AT&T Lines Go Dead (1990).** 75 million calls dropped (one line of buggy code in software upgrade).
- **Patriot missile defense fails (1991).** **28 soldiers dead, 100 injured** (rounding error in scud-detection).
- **Pentium floating point long-division bug (1993).** Cost: $475 million – flawed division table.
- **Ariane rocket explosion (1996).** Cost: $500 million – overflow in 64-bit to 16-bit conversion.
- **Y2K (1999).** Cost: $500 billion spent because year was stored as 2 digits to save space.
- **Mars Climate Orbiter Crash (1998).** Cost: $125 million lost due to metric to imperial units bug.
- **Tesla Self-Driving Car (2016). 1 dead**. Auto-pilot didn't "see" tractor-trailer.
- **Financial Disasters:** London Stock Exchange down due to single server bug (**2009**; billions of pounds of trading); Knight Capital computer glitch trigers stock sale (**2012**; 500 million lost and Knight's value drops by 75%).
- **Airline Disasters:**
  - AirFrance 447 2009, **228 dead**: pitot-tube failure feeds inconsistent data to programs which then panic pilot.
  - Spanair 5022, 2008, **154 dead**: malware virus.
  - AdamAir 574, 2007, **102 dead**: navigation system errors (and pilot errors).
  - KoreanAir 801, 1997, **228 dead**: ground proximity warning system bug.
  - AeroPerú 603, 1996, **70 dead**: altimeter failures.
  - Scottish RAF Chinook, 1994, **29 dead**: faulty test program
  - AirFrance 296, 1988, **3 dead**: altimeter bug.
  - IranAir 655, 1988, **290 dead**: shot down by US Aegis combat system (misidentified as attacking military plane).
  - KoreanAir 007, 1983, **269 dead**: autopilot took plane into Soviet airspace where it got shot down.
  - Boeing 737 Max, 2018,2019, **346 dead**: attack sensor + algorithm errors.
- **Software errors cost the U.S. $60 billion annually in rework, lost productivity and actual damages.**

Put effort to make *sure* your program works **fully** correctly **all** the time.