Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 et seq.)

<table>
<thead>
<tr>
<th>Number of Operations</th>
<th>Variable acted upon</th>
<th>Variables receiving results</th>
<th>Indication of change in the value on any Variable</th>
<th>Statement of Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IV₁ × IV₂</td>
<td>IV₁ × IV₃ × IV₄ × IV₅ × IV₆</td>
<td>- 2ⁿ</td>
<td>- 2ⁿ 2ⁿ 2ⁿ 2ⁿ 2ⁿ 2ⁿ</td>
</tr>
<tr>
<td>2</td>
<td>- IV₁ × IV₂</td>
<td>IV₁ × IV₃ × IV₄ × IV₅ × IV₆</td>
<td>2ⁿ-1</td>
<td>2ⁿ-1 2ⁿ-1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>3</td>
<td>+ IV₁ × IV₂</td>
<td>IV₁ × IV₃ × IV₄ × IV₅ × IV₆</td>
<td>2ⁿ+1</td>
<td>0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>4</td>
<td>+ IV₁ × IV₂</td>
<td>IV₁ × IV₃ × IV₄ × IV₅ × IV₆</td>
<td>2ⁿ-1</td>
<td>2ⁿ-1 2ⁿ-1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>5</td>
<td>+ IV₁ × IV₂</td>
<td>IV₁ × IV₃ × IV₄ × IV₅ × IV₆</td>
<td>2ⁿ-1</td>
<td>2ⁿ-1 2ⁿ-1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>6</td>
<td>+ IV₁ × IV₂</td>
<td>IV₁ × IV₃ × IV₄ × IV₅ × IV₆</td>
<td>2ⁿ-1</td>
<td>2ⁿ-1 2ⁿ-1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>7</td>
<td>- IV₁ × IV₂</td>
<td>IV₁ × IV₃ × IV₄ × IV₅ × IV₆</td>
<td>2ⁿ-1</td>
<td>2ⁿ-1 2ⁿ-1 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>
|                      | **Result Variables** |                          |                               | - 2ⁿ-1 2ⁿ-1 0 0 ...

Here follows a repetition of Operations thirteen to twenty-three.

(early computation path written by Ada Lovelace)
Extended New Year’s Review: Lectures 15–19
Alternation
Alternating Computations

Non-deterministic TMs:
- Accept if there is an accepting run.
- Used to define classes like NP

Complements of non-deterministic classes:
- Accept if all runs are accepting.
- Used to define classes like coNP

We have seen that existential and universal modes can also alternate:
- Players take turns in games
- Quantifiers may alternate in QBF

Is there a suitable Turing Machine model to capture this?
Alternating Turing Machines

Definition 14.1
An alternating Turing machine (ATM) $M = (Q, \Sigma, \Gamma, \delta, q_0)$ is a Turing machine with a non-deterministic transition function $\delta : Q \times \Gamma \rightarrow \mathfrak{P}(Q \times \Gamma \times \{L, R\})$ whose set of states is partitioned into existential and universal states:

$Q_\exists$: set of existential states \hspace{1cm} Q_U: set of universal states

- Configurations of ATMs are the same as for (N)TMs: tape(s) + state + head position
- A configuration can be universal or existential, depending on whether its state is universal or existential
- Possible transitions between configurations are defined as for NTMs
Alternating Turing Machines: Acceptance

Acceptance is defined recursively:

**Definition 14.2**

A configuration $C$ of an ATM $M$ is accepting if one of the following is true:

- $C$ is existential and some successor configuration of $C$ is accepting.
- $C$ is universal and all successor configurations of $C$ are accepting.

$M$ accepts a word $w$ if the start configuration on $w$ is accepting.
Alternating Turing Machines: Acceptance

Acceptance is defined recursively:

**Definition 14.2**

A configuration $C$ of an ATM $M$ is *accepting* if one of the following is true:
- $C$ is existential and some successor configuration of $C$ is accepting.
- $C$ is universal and all successor configurations of $C$ are accepting.

$M$ accepts a word $w$ if the start configuration on $w$ is accepting.

**Note:** configurations with no successor are the base case, since we have:
- An existential configuration without any successor configurations is rejecting.
- A universal configuration without any successor configurations is accepting.

Hence we don’t need to specify accepting or rejecting states explicitly.
Nondeterminism and Parallelism

ATMs can be seen as a generalisation of non-deterministic TMs:

An NTM is an ATM where all states are existential (besides the single accepting state, which is always universal according to our definition).
Nondeterminism and Parallelism

ATMs can be seen as a generalisation of non-deterministic TMs:

An NTM is an ATM where all states are existential (besides the single accepting state, which is always universal according to our definition).

ATMs can be seen as a model of parallel computation:

In every step, fork the current process to create sub-processes that explore each possible transition in parallel

- for universal states, combine the results of sub-processes with AND
- for existential states, combine the results of sub-processes with OR

Alternative view: an ATM accepts if its computation tree, considered as an AND-OR tree, evaluates to TRUE
Example: Alternating Algorithm for MinFormula

**MinFormula**

*Input:* A propositional formula \( \varphi \).

*Problem:* Is \( \varphi \) the shortest formula that is satisfied by the same assignments as \( \varphi \)?
Example: Alternating Algorithm for $\text{MinFormula}$

$\text{MinFormula}$

\text{Input: } A propositional formula $\varphi$.
\text{Problem: } Is $\varphi$ the shortest formula that is satisfied by the same assignments as $\varphi$?

$\text{MinFormula}$ can be solved by an alternating algorithm:

\begin{verbatim}
01 MinFormula(formula $\varphi$) :
02 universally choose $\psi := \text{formula shorter than } \varphi$
03 exist. guess $\mathcal{I} := \text{assignment for variables in } \varphi$
04 if $\varphi^{\mathcal{I}} = \psi^{\mathcal{I}}$ :
05    return FALSE
06 else :
07    return TRUE
\end{verbatim}
Example: Alternating Algorithm for Geography

```
01 ALTGEOGRAPHY(directed graph G, start node s) :
02   Visited := \{s\}  // visited nodes
03   cur := s  // current node
04   while TRUE :
05     // existential move:
06     if all successors of cur are in Visited:
07       return FALSE
08     existentially guess cur := unvisited successor of cur
09     Visited := Visited \cup \{cur\}
10     // universal move:
11     if all successors of cur are in Visited:
12       return TRUE
13     universally choose cur := unvisited successor of cur
14     Visited := Visited \cup \{cur\}
```
Time and Space Bounded ATMs

As before, time and space bounds apply to any computation path in the computation tree.

Definition 14.3
Let $M$ be an alternating Turing machine and let $f : \mathbb{N} \to \mathbb{R}^+$ be a function.

- $M$ is $f$-time bounded if it halts on every input $w \in \Sigma^*$ and on every computation path after $\leq f(|w|)$ steps.

- $M$ is $f$-space bounded if it halts on every input $w \in \Sigma^*$ and on every computation path using $\leq f(|w|)$ cells on its tapes.

(Here we typically assume that Turing machines have a separate input tape that we do not count in measuring space complexity.)
Defining Alternating Complexity Classes

Definition 14.4

Let $f : \mathbb{N} \rightarrow \mathbb{R}^+$ be a function.

- $\text{ATime}(f(n))$ is the class of all languages $\mathcal{L}$ for which there is an $O(f(n))$-time bounded alternating Turing machine deciding $\mathcal{L}$, for some $k \geq 1$.

- $\text{ASpace}(f(n))$ is the class of all languages $\mathcal{L}$ for which there is an $O(f(n))$-space bounded alternating Turing machine deciding $\mathcal{L}$.
Common Alternating Complexity Classes

\[ \text{AP} = \text{APTime} = \bigcup_{d \geq 1} \text{ATime}(n^d) \]
alternating polynomial time

\[ \text{AExp} = \text{AExpTime} = \bigcup_{d \geq 1} \text{ATime}(2^{n^d}) \]
alternating exponential time

\[ \text{A2Exp} = \text{A2ExpTime} = \bigcup_{d \geq 1} \text{ATime}(2^{2n^d}) \]
alt. double-exponential time

\[ \text{AL} = \text{ALogSpace} = \text{ASpace}(\log n) \]
alternating logarithmic space

\[ \text{APSpace} = \bigcup_{d \geq 1} \text{ASpace}(n^d) \]
alternating polynomial space

\[ \text{AExpSpace} = \bigcup_{d \geq 1} \text{ASpace}(2^{n^d}) \]
alternating exponential space

Example: \text{GEOGRAPHY} \in \text{APTime}
Alternating Complexity Classes: Basic Properties

Nondeterminism: ATMs can do everything that the corresponding NTMs can do, e.g., $\text{NP} \subseteq \text{APTIME}$
Alternating Complexity Classes: Basic Properties

**Nondeterminism:** ATMs can do everything that the corresponding NTMs can do, e.g., $\text{NP} \subseteq \text{APTime}$

**Reducions:** Polynomial many-one reductions can be used to show membership in many alternating complexity classes, e.g., if $\mathcal{L} \in \text{APTime}$ and $\mathcal{L}' \leq_p \mathcal{L}$ then $\mathcal{L}' \in \text{APTime}$. 
Alternating Complexity Classes: Basic Properties

Nondeterminism: ATMs can do everything that the corresponding NTMs can do, e.g., $\text{NP} \subseteq \text{APTime}$

Reductions: Polynomial many-one reductions can be used to show membership in many alternating complexity classes, e.g., if $L \in \text{APTime}$ and $L' \leq_p L$ then $L' \in \text{APTime}$.

In particular: $\text{PSPACE} \subseteq \text{APTime}$ (since Geography $\in \text{APTime}$)
Alternating Complexity Classes: Basic Properties

Nondeterminism: ATMs can do everything that the corresponding NTMs can do, e.g., $\text{NP} \subseteq \text{APTime}$

Reductions: Polynomial many-one reductions can be used to show membership in many alternating complexity classes, e.g., if $L \in \text{APTime}$ and $L' \leq_p L$ then $L' \in \text{APTime}$.

In particular: $\text{PSPACE} \subseteq \text{APTime}$ (since Geography $\in \text{APTime}$)

Complementation: ATMs are easily complemented:

- Let $M$ be an ATM accepting language $L(M)$
- Let $M'$ be obtained from $M$ by swapping existential and universal states
- Then $L(M') = \overline{L(M)}$

For alternating algorithms this means: (1) negate all return values, (2) swap universal and existential branching points
Example: Complement of $\text{MinFormula}$

Original algorithm:

01 $\text{MinFormula}(\text{formula } \varphi)$ :
02 universally choose $\psi := \text{formula shorter than } \varphi$
03 exist. guess $I := \text{assignment for variables in } \varphi$
04 if $\varphi_I = \psi_I$ :
05 return FALSE
06 else :
07 return TRUE

Complemented algorithm:

01 $\text{ComplMinFormula}(\text{formula } \varphi)$ :
02 existentially guess $\psi := \text{formula shorter than } \varphi$
03 univ. choose $I := \text{assignment for variables in } \varphi$
04 if $\varphi_I = \psi_I$ :
05 return TRUE
06 else :
07 return FALSE