

Complexity Theory Alternation

Daniel Borchmann, Markus Kröttsch

Computational Logic

2016-01-05



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

Number of Operation. Name of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data.												Working Variables.					Result Variables.							
					v ₁	v ₂	v ₃	v ₄	v ₅	v ₆	v ₇	v ₈	v ₉	v ₁₀	v ₁₁	v ₁₂	v ₁₃	v ₁₄	v ₁₅	v ₁₆	v ₁₇	v ₁₈	v ₁₉	v ₂₀	v ₂₁	v ₂₂	v ₂₃	v ₂₄	v ₂₅
					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	x	v ₂ × v ₁	v ₄ , v ₅ , v ₆	{v ₄ = v ₂ v ₅ = v ₂ v ₆ = v ₂ }	= 2n ...	2	n	2n	2n	2n																			
2	-	v ₄ - v ₂	v _{11}}	{v ₁₁ = v ₄ v ₁₁ = v ₄ }	= 2n - 1 ...	1		2n - 1																					
3	+	v ₄ + v ₂	v ₃	{v ₃ = v ₄ v ₃ = v ₄ }	= 2n + 1 ...	1			2n + 1																				
4	+	v ₄ + v ₂	v _{11}}	{v ₁₁ = v ₄ v ₁₁ = v ₄ }	= 2n - 1 ...	0	0																						
5	+	v ₄ + v ₂	v _{11}}	{v ₁₁ = v ₄ v ₁₁ = v ₄ }	= 2n - 1 ...	2																							
6	-	v ₄ - v ₂	v _{13}}	{v ₁₃ = v ₄ v ₁₃ = v ₄ }	= 1/2 * 2n - 1 ...																								
7	-	v ₄ - v ₂	v _{13}}	{v ₁₃ = v ₄ v ₁₃ = v ₄ }	= n - 1 ...	1	n																						
8	+	v ₂ + v ₂	v ₇	{v ₇ = v ₂ v ₇ = v ₂ }	= 2 + 0 = 2 ...	2																							
9	+	v ₂ + v ₂	v _{11}}	{v ₁₁ = v ₂ v ₁₁ = v ₂ }	= 2 = A ₁ ...					2n	2																		
10	x	v ₂ × v ₁₁	v _{12}}	{v ₁₂ = v ₂ v ₁₂ = v ₂ }	= B ₁ * 2 = B ₁ A ₁ ...																								
11	+	v ₁₂ + v ₁₁	v _{13}}	{v ₁₃ = v ₁₂ v ₁₃ = v ₁₂ }	= 1/2 * 2n - 1 + B ₁ * 2n/2 ...																								
12	-	v ₁₂ - v ₁₁	v _{13}}	{v ₁₃ = v ₁₂ v ₁₃ = v ₁₂ }	= n - 2 ...	1																							
13	-	v ₂ - v ₂	v ₆	{v ₆ = v ₂ v ₆ = v ₂ }	= 2n - 1 ...	1																							
14	+	v ₂ + v ₂	v ₇	{v ₇ = v ₂ v ₇ = v ₂ }	= 2 + 1 = 3 ...	1					3																		
15	+	v ₂ + v ₂	v ₆	{v ₆ = v ₂ v ₆ = v ₂ }	= 2n - 1 ...	1					2n - 1	3	2n - 1	3															
16	x	v ₆ × v ₁₁	v _{13}}	{v ₁₃ = v ₆ v ₁₃ = v ₆ }	= 2n * 2n - 1 ...																								
17	-	v ₆ - v ₁₁	v ₆	{v ₆ = v ₆ v ₆ = v ₆ }	= 2n - 2 ...	1																							
18	+	v ₂ + v ₂	v ₇	{v ₇ = v ₂ v ₇ = v ₂ }	= 3 + 1 = 4 ...	1					4																		
19	+	v ₆ + v ₂	v ₆	{v ₆ = v ₆ v ₆ = v ₆ }	= 2n - 2 ...							4		2n - 2	4														
20	x	v ₆ × v ₁₁	v _{13}}	{v ₁₃ = v ₆ v ₁₃ = v ₆ }	= 2n * 2n - 1 ...																								
21	x	v ₆ × v ₁₁	v _{13}}	{v ₁₃ = v ₆ v ₁₃ = v ₆ }	= B ₂ * 2 * 2n - 1 * 2n - 2 = B ₂ A ₂ ...																								
22	+	v ₆ + v ₂	v _{13}}	{v ₁₃ = v ₆ v ₁₃ = v ₆ }	= A ₂ + B ₁ A ₁ + B ₂ A ₂ ...																								
23	-	v ₆ - v ₁₁	v _{13}}	{v ₁₃ = v ₆ v ₁₃ = v ₆ }	= n - 3 ...	1																							
Here follows a repetition of Operations thirteen to twenty-three.																													
24	+	v ₁₃ + v ₆	v _{13}}	{v ₁₃ = v ₁₃ v ₁₃ = v ₁₃ }	= B ₂ ...																								
25	+	v ₁ + v ₆	v _{5}}	{v ₅ = v ₁ v ₅ = v ₁ }	= n + 1 = 4 + 1 = 5 ...	1	n + 1																						

(early computation path written by Ada Lovelace)

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) $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0)$ 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

Q_{\forall} : 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 \mathcal{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.

\mathcal{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).

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 φ .

Problem: Is φ the shortest formula that is satisfied by the same assignments as φ ?

MINFORMULA can be solved by an alternating algorithm:

```

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

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 \mathcal{M} be an alternating Turing machine and let $f : \mathbb{N} \rightarrow \mathbb{R}^+$ be a function.

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

- ▶ $ATIME(f(n))$ 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 $k \geq 1$.
- ▶ $ASPACE(f(n))$ 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

$AP = APTIME = \bigcup_{d \geq 1} ATIME(n^d)$ alternating polynomial time

$AEXP = AEXPTIME = \bigcup_{d \geq 1} ATIME(2^{n^d})$ alternating exponential time

$A2EXP = A2EXPTIME = \bigcup_{d \geq 1} ATIME(2^{2^{n^d}})$ alt. double-exponential time

$AL = ALOGSPACE = ASPACE(\log n)$ alternating logarithmic space

$APSPACE = \bigcup_{d \geq 1} ASPACE(n^d)$ alternating polynomial space

$AEXPSPACE = \bigcup_{d \geq 1} ASPACE(2^{n^d})$ alternating exponential space

Example: $GEOGRAPHY \in APTIME$

Alternating Complexity Classes: Basic Properties

Nondeterminism: ATMs can do everything that the corresponding NTMs can do, e.g., $NP \subseteq APTIME$

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

In particular: $PSPACE \subseteq APTIME$ (since $GEOGRAPHY \in APTIME$)

Complementation: ATMs are easily complemented:

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

For alternating algorithms this means: (1) negate all return values, (2) swap universal and existential branching points

Example: Complement of MINFORMULA

Original algorithm:

```

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

```

Complemented algorithm:

```

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

```