Simultaneous Resource Classes

Def: ATIME-ALT\([t(n), a(n)]\) is the set of problems solved by alternating Turing machines in time \(O(t(n))\), while making at most \(a(n)\) alternations between existential and universal states, starting with existential.

Example: \(\text{ATIME-ALT}[t(n), 0] = \text{NTIME}[t(n)]\)

Similarly, define the simultaneous classes:

\[\text{ASPACE-TIME}[s(n), t(n)], \quad \text{ASPACE-ALT}[s(n), a(n)]\]
Define the LOGTIME hierarchy (LH) and the PTIME hierarchy (PH) as follows:

\[
\begin{align*}
\text{LH} &= \text{ATIME-ALT}[\log n, O(1)] \\
\text{PH} &= \text{ATIME-ALT}[n^{O(1)}, O(1)]
\end{align*}
\]

**Thm:** \( \text{PH} = \text{SO} \)

**Proof:** [idea] Follows from Fagin’s Thm: \( \text{NP} = \text{SO}\exists \).

**Fact:** \( \text{LH} = \text{FO} \)
CRAM\([t(n)] = \text{CRCW-PRAM-TIME}[t(n)]-\text{HARD}[n^{O(1)}]\)
synchronous, concurrent, \(n^{O(1)}\) processors and memory

priority write: lowest number processor wins conflict

common write: no conflicts allowed
**Thm:** For all $t(n)$, $\text{CRAM}[t(n)] = \text{AC}[t(n)]$.

**Proof:** (Sketch)

$\text{CRAM}[1] \subseteq \text{AC}[1] = \text{AC}^0 = \text{FO}$:

Only the global memory is tricky: all else is sequential.

$B(i, w, b)$ means that bit $i$ of global memory word $w$ is $b$.

$$B'(i, w, b) \equiv B(i, w, b) \land \forall p \left( \text{“}p \text{ didn’t just write into word } w\text{”} \right)$$

$$\lor \exists p \left( \text{“}p \text{ just wrote into word } w\text{”} \land R(p, i, b) \right)$$
CRAM[1] ⊃ FO:

∀x(α(x))

1. w := 1
2. Each processor P_i in parallel, do {
3. if (¬α(i)) then Write(0,w)
4. }

∃x(α(x))

1. w := 0
2. Each processor P_i in parallel, do {
3. if (α(i)) then Write(1,w)
4. }
**Def:** sAC$^1$ (semi-unbounded sAC$^1$) is the subset of AC$^1$ where the and-gates are binary and only the or-gates are unbounded.

**Fact:** [Ruzzo] sAC$^1 = \log(CFL) = \text{FO}(CFL) = \{ S \mid \exists \text{CFL} \ C \ (S \leq C) \}

**Thm:** NC$^1 \subseteq L \subseteq \text{NL} \subseteq \text{sAC}^1 \subseteq \text{AC}^1

**Proof:**
Alternation as Parallelism

**Fact:** [Ruzzo and Tompa] For \( t(n) \geq \log n \),

\[
\begin{align*}
\text{ASPACE-TIME}[\log n, t(n)] &= \text{NC}[t(n)] \\
\text{ASPACE-ALT}[\log n, t(n)] &= \text{AC}[t(n)]
\end{align*}
\]

**Cor:** \( \text{ATIME}[\log n] = \text{NC}^1 \)
\begin{align*}
\text{ASPACE-TIME}[\log n, t(n)] &= \text{NC}[t(n)] \\
\text{ASPACE-ALT}[\log n, t(n)] &= \text{AC}[t(n)]
\end{align*}

**Proof:** (sketches)

The computation graph of an ASPACE-TIME\([\log n, t(n)]\) machine on an input of size \(n\) is an \(\text{NC}[t(n)]\) circuit.

The computation graph of an ASPACE-ALT\([\log n, t(n)]\) machine is a series of \(t(n)\) computation graphs of NL or co-NL machines.

We can modify the ASPACE-ALT\([\log n, t(n)]\) machine to make a series of alternating guesses: \(\text{ID}_1, \text{ID}_2, \ldots \text{ID}_{t(n)}\) of the endpoints of each of these NL or co-NL computations, and **check only once** that \(\text{ID}_{i+1}\) follows correctly from \(\text{ID}_i\).

Since \(\text{NL} \subseteq \text{AC}^1\), the whole thing is an \(\text{AC}[t(n) + \log n] = \text{AC}[t(n)]\) circuit. \(\square\)