Mitosis in Computational Complexity

Christian Glaßer\textsuperscript{1}, A. Pavan\textsuperscript{2,*}, Alan L. Selman\textsuperscript{3,**}, and Liyu Zhang\textsuperscript{3}

\textsuperscript{1} Universität Würzburg
glasser@informatik.uni-wuerzburg.de
\textsuperscript{2} Iowa State University
pavan@cs.iastate.edu
\textsuperscript{3} University at Buffalo
\{selman, lzhang7\}@cse.buffalo.edu

Abstract. This expository paper describes some of the results of two recent research papers \cite{GOP+05, GPSZ05}. The first of these papers proves that every NP-complete set is many-one autoreducible. The second paper proves that every many-one autoreducible set is many-one mitotic. It follows immediately that every NP-complete set is many-one mitotic. Hence, we have the compelling result that every NP-complete set \( A \) splits into two NP-complete sets \( A_1 \) and \( A_2 \).

1 Autoreducibility

We begin with the notion of autoreducibility. Trakhtenbrot \cite{Tra70} defined a set \( A \) to be \textit{autoreducible} if it can be reduced to itself by a Turing machine that does not ask its own input to the oracle. This means there is an oracle Turing machine \( M \) such that \( A = L(M^A) \) and \( M \) on input \( x \) never queries \( x \). Ladner \cite{Lad73} showed that there exist Turing-complete recursively enumerable sets that are not autoreducible. We are interested in the polynomial-time variant of autoreducibility, introduced by Ambos-Spies \cite{AS84}, where we require the oracle Turing machine to run in polynomial time. Henceforth, by “autoreducible” we mean “polynomial-time autoreducible.”

The question of whether complete sets for various complexity classes are autoreducible has been studied extensively \cite{Yao90, BF92, BFvMT00}. Beigel and Feigenbaum \cite{BF92} showed that Turing-complete sets for the classes that form the polynomial-time hierarchy are autoreducible. In particular, all Turing-complete sets for NP are autoreducible. Buhrman et al. \cite{BFvMT00} showed that Turing-complete sets for EXP and \( \Delta^p \) are autoreducible, whereas there exists a Turing-complete set for EESPACE that is not Turing autoreducible. They showed that answering questions about autoreducibility of intermediate classes results in interesting separation results.

Researchers have studied autoreducibility with various polynomial-time reducibilities. Regarding NP, Buhrman et al. \cite{BFvMT00} showed that all truth-table-complete sets for NP are probabilistic truth-table autoreducible. Thus, all NP-complete sets are probabilistic truth-table autoreducible.

* Research supported in part by NSF grants CCR-0344817 and CCF-0430807.
** Research supported in part by NSF grant CCR-0307077.
A set $A$ is *polynomial-time m-autoreducible* (m-autoreducible, for short) if $A \leq^p_m A$ via a polynomial-time computable reduction function $f$ such that for all $x$, $f(x) \neq x$. Note that m-autoreducibility is a strong form of autoreducibility. If a set is m-autoreducible, then it is T-autoreducible also. Buhrman and Torenvliet [BT94] asked whether all NP-complete sets are m-autoreducible and whether all PSPACE-complete sets are m-autoreducible. Glasser et al. [GOP+05] resolve these questions positively. A set $L$ is *nontrivial* if $|L| > 1$ and $|\overline{L}| > 1$. The proof that all nontrivial NP-complete sets are m-autoreducible is interesting, for it uses the left set technique of Ogihara and Watanabe [OW91].

Let $L$ belong to the class NP and let $M$ be a polynomial-time-bounded non-deterministic Turing machine that accepts $L$. For a suitable polynomial $p$, we can assume that all computation paths $v$ on input $x$ have length $p(|x|)$. Let

$$\text{Left}(L) = \{(x,u) \mid |u| = p(|x|) \text{ and } \exists v, |v| = |u|, \text{such that } u \leq v \text{ and } M(x) \text{ accepts along path } v\}.$$ 

Notice that Left($L$) belongs to the class NP. So if $L$ is NP-complete, then there is a polynomial-time-computable reduction $f$ from Left($L$) to $L$.

**Theorem 1.** All nontrivial NP-complete sets are m-autoreducible.

**Proof.** Let $L$ be NP-complete. As we just described, let $M$ be an NP-machine that accepts $L$, let $p$ be a polynomial so that all computation paths of $M$ on an input $x$ have length $p(|x|)$, and let $f$ be a polynomial-time-computable reduction from Left($L$) to $L$. Since $L$ is nontrivial, let $y_1, y_2 \in L$ and $\overline{y}_1, \overline{y}_2 \in \overline{L}$.

The following algorithm defines a function $g$ to be an m-autoreduction for $L$:

Let $x$ be an input, and define $n = |x|$ and $m = p(|x|)$.

```plaintext
1 if f((x,0^n)) \neq x then output f((x,0^n))
2 if f((x,1^n)) = x then
3     if M(x) accepts along 1^n then
4         output a string from \{y_1, y_2\} - \{x\}
5     else
6         output a string from \{\overline{y}_1, \overline{y}_2\} - \{x\}
7     endif
8 endif
9 // here f((x,0^n)) = x \neq f((x,1^n))
10 determine z of length m such that f((x,z)) = x \neq f((x,z + 1))
11 if M(x) accepts along z then output a string from \{y_1, y_2\} - \{x\}
12 else output f((x,z + 1))
```

Step 10 is accomplished by an easy binary search algorithm: Start with $z_1 := 0^m$ and $z_2 := 1^m$. Let $z'$ be the middle element between $z_1$ and $z_2$. If $f(z') = x$ then $z_1 := z'$ else $z_2 := z'$. Again, choose the middle element between $z_1$ and $z_2$, and so on. This shows that $g$ is computable in polynomial time. Clearly, $g(x) \neq x$, so it remains to show that $L \leq^p_m L$ via $g$.

If the algorithm stops in step 1, then

$$x \in L \iff \langle x,0^m \rangle \in \text{Left}(L) \iff g(x) = f(\langle x,0^m \rangle) \in L.$$