The Self-Reducibility Technique

Alan Beadle  Komail Dharsee  David Fink
Gene Kim  Joel Kottas
Xiangru Lian  Zahra Razavi

November 2016
The Pruning Technique

Part I

The Pruning Technique
Definitions

- **Self-reducibility:**
  \( L \) is self reducible if there is a polynomial-time oracle Turing Machine \( T^L \) such that \( L = L(T^L) \), and \( T^L \) on input \( x \) queries its oracle only with strings of length \( < |x| \).
Definitions

- Self-reducibility: 
  \( L \) is self reducible if there is a polynomial-time oracle Turing Machine \( T^L \) such that \( L = L(T^L) \), and \( T^L \) on input \( x \) queries its oracle only with strings of length \( < |x| \).

- Examples:
  - SAT (a formula is satisfiable iff it is satisfiable with some variable \( x \) set to \( T \) or with \( x \) set to \( F \)).
Definitions

- **Self-reducibility:**
  $L$ is self reducible if there is a polynomial-time oracle Turing Machine $T^L$ such that $L = L(T^L)$, and $T^L$ on input $x$ queries its oracle only with strings of length $< |x|$.

- **Examples:**
  - SAT (a formula is satisfiable iff it is satisfiable with some variable $x$ set to $T$ or with $x$ set to $F$).
  - Anything in $P$
Definitions (cont.)

- Tally set:
  
  $T$ is a tally set if $T \subseteq 1^*$
Tally set:
$T$ is a tally set if $T \subseteq 1^*$

Examples: $\emptyset, 1^*, \{1^p \mid L(N_p) \in P\}$
Census function:
For an set $A$, $c_A(n) = |\{x \in A \mid |x| \leq n\}|$
Definitions (cont.)

- **Census function:**
  For an set $A$, $c_A(n) = \| \{ x \in A \mid |x| \leq n \} \|$.

- **Examples:**
  - $c_{\emptyset}(n) = 0$.
Definitions (cont.)

- Census function:
  For an set $A$, $c_A(n) = \| \{ x \in A \mid |x| \leq n \} \|$.

- Examples:
  - $c_{\emptyset}(n) = 0$
  - $c_{SAT}$ is exponential

The Self-Reducibility Technique 5/34
Definitions (cont.)

- Census function:
  For an set $A$, $c_A(n) = |\{ x \in A \mid |x| \leq n \}|$

- Examples:
  - $c_{\emptyset}(n) = 0$
  - $c_{SAT}$ is exponential

- Density:
  Rate of growth of census function
Definitions (cont.)

- Finite sets have a bounded census function (if $||A|| < \infty$, $\exists M \forall n \ c_A(n) \leq M$)

- Sparse sets: $A$ is sparse if there is a polynomial $p$ such that $c_A \leq p$

- Examples:
  - Any finite set
  - $\{n | n \text{ is a power of 2}\}$
  - $A$ such that $A$ contains the first $n$ lexicographically first strings of length $n$
Finite sets have a bounded census function (if $|A| < \infty$, $\exists M \forall n c_A(n) \leq M$)

Sparse sets:
$A$ is sparse if there is a polynomial $p$ such that $c_A \leq p$
Definitions (cont.)

- Finite sets have a bounded census function (if $|A| < \infty$, $\exists M \forall n \ c_A(n) \leq M$)

- Sparse sets:
  $A$ is sparse if there is a polynomial $p$ such that $c_A \leq p$

- Examples:
  - Any finite set
Definitions (cont.)

- Finite sets have a bounded census function (if $||A|| < \infty$, $\exists M \forall n \ c_A(n) \leq M$)
- Sparse sets:
  $A$ is sparse if there is a polynomial $p$ such that $c_A \leq p$
- Examples:
  - Any finite set
  - $\{n \mid n \text{ is a power of } 2\}$
Definitions (cont.)

- Finite sets have a bounded census function (if $|A| < \infty$, $\exists M \forall n c_A(n) \leq M$)
- Sparse sets:
  $A$ is sparse if there is a polynomial $p$ such that $c_A \leq p$
- Examples:
  - Any finite set
  - $\{n \mid n$ is a power of 2$\}$
  - $A$ such that $A$ contains the first $n$ lexicographically first strings of length $n$
Definitions (cont.)

- Dense sets:
  A is dense if there is a linear-exponential function $f$ and an $N$ such that, $\forall n > N$, $c_A(n) \geq f(n)$
Definitions (cont.)

- **Dense sets:**
  \( A \) is dense if there is a linear-exponential function \( f \) and an \( N \) such that, \( \forall n > N, c_A(n) \geq f(n) \)

- **Linear-exponential:**
  \( f \) is linear-exponential if \( f(n) \) is of the form \( BC^n \), where \( B > 0 \) and \( C > 1 \).
Dense sets:
A is dense if there is a linear-exponential function $f$ and an $N$ such that, $\forall n > N, c_A(n) \geq f(n)$

Linear-exponential:
$f$ is linear-exponential if $f(n)$ is of the form $BC^n$, where $B > 0$ and $C > 1$.

Examples:
- $\Sigma^*$
Definitions (cont.)

- **Dense sets:**
  A is dense if there is a linear-exponential function $f$ and an $N$ such that, $\forall n > N$, $c_A(n) \geq f(n)$

- **Linear-exponential:**
  $f$ is linear-exponential if $f(n)$ is of the form $BC^n$, where $B > 0$ and $C > 1$.

- **Examples:**
  - $\Sigma^*$
  - SAT
Equivalent definition of sparse sets:

A is sparse if there is a polynomial \( p \) such that

\[
\forall n \ | \ | \{ x \in A \ | \ |x| = n \} | < p(n)
\]
Equivalence of Definitions of Sparse Sets

Theorem
Let $b_A(n) = || \{ x \in A | |x| = n \} ||$. Then there is a polynomial $q$ such that $b_A \leq q$ if and only if $A$ is sparse.
The Pruning Technique

Equivalence of Definitions of Sparse Sets

**Theorem**

Let $b_A(n) = || \{ x \in A \mid |x| = n \} ||$. Then there is a polynomial $q$ such that $b_A \leq q$ if and only if $A$ is sparse.

**Proof.**

Suppose $A$ is sparse, so $p \geq c_A(n)$. Then $q = p$ works, since $b_A \leq c_A$. 

Theorem

Let $b_A(n) = \| \{ x \in A \mid |x| = n \} \|$. Then there is a polynomial $q$ such that $b_A \leq q$ if and only if $A$ is sparse.

Proof.

Suppose $A$ is sparse, so $p \geq c_A(n)$. Then $q = p$ works, since $b_A \leq c_A$.

Conversely, let $q \geq b_A$. We have $q(n) \leq n^d + d$ for some positive integer $d$. Then $p(n) = n^{d+1} + n^d + dn + d$ works, since

$$c_A(n) = \sum_{k=0}^{n} b_A(k) \leq \sum_{k=0}^{n} q(k) \leq \sum_{k=0}^{n} k^d + d \leq n^{d+1} + n^d + dn + d,$$

so $A$ is sparse.
Theorem 1.2

- Theorem

If there is a tally set that is ≤\text{m}^\text{p}-hard for NP, then P=NP
Theorem 1.2

- **Theorem**
  
  *If there is a tally set that is \( \leq^p_m \)-hard for \( NP \), then \( P=NP \)*

- **Corollary (Theorem 1.3)**
  
  *If there is a tally set that is NP-complete, then \( P=NP \)*

Note: They are actually "iff" theorems, since the converses of both the Theorem and the Corollary are true for the set \( \{1\} \) which is both NP-complete and coNP-complete.
The Pruning Technique

Theorem 1.2

- **Theorem**
  
  *If there is a tally set that is $\leq^p_m$-hard for NP, then $P=NP$*

- **Corollary (Theorem 1.3)**
  
  *If there is a tally set that is NP-complete, then $P=NP$*

- **Note:** They are actually ”iff” theorems, since the converses of both the Theorem and the Corollary are true for the set $\{1\}$ which is both NP-complete and coNP-complete
Proof of Theorem 1.2

- Assume $T$ is a tally set that is $\leq^p_m$-hard for NP $\implies$ $SAT \leq^p_m T$

So, $\exists$ deterministic poly-time function $\sigma(.)$ which maps $SAT$ to $T$,

$$x \in SAT \iff \sigma(x) \in T$$
Proof of Theorem 1.2

- Assume \( T \) is a tally set that is \( \leq^p_m \)-hard for \( \text{NP} \) \( \implies \) \( \text{SAT} \leq^p_m T \)

So, \( \exists \) deterministic poly-time function \( \sigma(.) \) which maps \( \text{SAT} \) to \( T \),

\[
x \in \text{SAT} \iff \sigma(x) \in T
\]

- \( \sigma(.) \) is computable by a deterministic poly-time Turing Machine, so:

\[
\exists k \ (\forall x) [|\sigma(x)| \leq |x|^k + k]
\]
Proof of Theorem 1.2

- **Claim:**
  We can propose a deterministic polynomial-time algorithm for SAT.
Proof of Theorem 1.2

Claim:
We can propose a deterministic polynomial-time algorithm for SAT.

Assume that the boolean formula, \( F \), is the input to SAT,

\[
F(v_1, v_2, \ldots, v_m)
\]

We go through \((m + 1)\) stages and at each stage we get rid of one of the variables by assigning both values True and False to it.

At the very last stage we will not have any variable and can simply decide the satisfiability.
The Pruning Technique

Theorem 1.2

$F$
The Pruning Technique

Theorem 1.2

\[ F \]

\[ F[v_1 = True] \quad F[v_1 = False] \]
Theorem 1.2

\[ F[v_1 = \text{True}] \]
\[ F[v_1 = \text{False}, v_2 = \text{False}] \]
\[ F[v_1 = \text{True}, v_2 = \text{True}] \]
\[ F[v_1 = \text{False}, v_2 = \text{True}] \]
The Pruning Technique

Theorem 1.2

\[ F[v_1 = True] \]

\[ F[v_1 = False] \]

\[ F[v_1 = True, v_2 = True] \]

\[ F[v_1 = False, v_2 = False] \]

\[ F[v_1 = False, v_2 = True] \]

\[ \ldots \]

\[ \ldots \]

\[ \ldots \]

\[ F[v_1 = T, v_2 = T, \ldots, v_m = T] \]
The Pruning Technique

Theorem 1.2

\[ F[v_1 = True] \]
\[ F[v_1 = False] \]
\[ F[v_1 = True, v_2 = True] \]
\[ F[v_1 = False, v_2 = False] \]
\[ F[v_1 = False, v_2 = True] \]

\[ F[v_1 = T, v_2 = T, ..., v_m = T] \]
At stage $i$:

Assume we have $l$ formulae as the input (which are the formulae resulting from stage $i - 1$)

$$Z = \{G_1, G_2, ..., G_l\}$$
The Pruning Technique

Proof of Theorem 1.2

▶ At stage \( i \):
Assume we have \( l \) formulae as the input (which are the formulae resulting from stage \( i - 1 \))

\[
Z = \{ G_1, G_2, ..., G_l \}
\]

▶ \( Z_1 = \{ G_1(v_i = True), G_2(v_i = True), ..., G_l(v_i = True), G_1(v_i = False), G_2(v_i = False), ..., G_l(v_i = False) \} \)

\( Z' = \{ \} \) (output formulae from Stage \( i \))
Proof of Theorem 1.2

- **At stage $i$:**
  Assume we have $l$ formulae as the input (which are the formulae resulting from stage $i - 1$)
  \[ Z = \{G_1, G_2, ..., G_l\} \]

- $Z_1 = \{G_1(v_i = True), G_2(v_i = True), ..., G_l(v_i = True), G_1(v_i = False), G_2(v_i = False), ..., G_l(v_i = False)\}$
  \[ Z' = \{\} \quad \text{(output formulae from Stage } i) \]

- For each formula $G$ in $Z$, compute $\sigma(G)$,
  Add $G$ to $Z'$ only if:
  (i) $\sigma(G) \in 1^*$, and
  (ii) no formula $H$ already exists in $Z'$ s.t. $\sigma(G) = \sigma(H)$
Proof of Theorem 1.2

Claim 1:
This algorithm works!

Claim 2:
This algorithm is polynomial-time!
Proof of Theorem 1.2

Claim 1:

\[ F \text{ is satisfiable} \quad \iff \quad \text{some member of the formula set output by stage } m \text{ is satisfiable} \]
Proof of Theorem 1.2

- **First:** The satisfiability of $Z$ is equivalent to the satisfiability of $Z_1$

\[ G \text{ is satisfiable} \iff (G_T \text{ is satisfiable}) \lor (G_F \text{ is satisfiable}) \]
First: The satisfiability of $Z$ is equivalent to the satisfiability of $Z_1$.

$G$ is satisfiable

$\iff$

$(G_T \text{ is satisfiable}) \lor (G_F \text{ is satisfiable})$

(Notation hint: $G_T = G[v = True]$ and $G_F = G[v = False]$ for the currently selected variable $v$)
Proof of Theorem 1.2

- **Second:** The satisfiability of $Z_1$ is equivalent to the satisfiability of $Z'$ (i.e. (i) and (ii) are harmless)

  $T \subseteq 1^*$, so $\sigma(G) \notin 1^* \implies \sigma(G) \notin T$
  
  $\implies G$ is not satisfiable

  so we can throw away $G$
Second: The satisfiability of $Z_1$ is equivalent to the satisfiability of $Z'$ (i.e. (i) and (ii) are harmless)

$T \subseteq 1^*$, so $\sigma(G) \notin 1^* \implies \sigma(G) \notin T \implies G$ is not satisfiable

so we can throw away $G$

if $\exists H$ s.t. $\sigma(G) = \sigma(H)$

$$\sigma(G) \in T \iff G \in SAT$$

$$\sigma(H) \in T \iff H \in SAT$$

then

$$G \in SAT \iff H \in SAT$$

so we can just throw away one of them.
Claim 2:

The algorithm is polynomial-time
Proof of Theorem 1.2

* Assume $|F| = p$ \hfill (1)
Proof of Theorem 1.2

* Assume $|F| = p$ \hspace{1cm} (1)
* Length of formulae will not be increased at each stage \hspace{1cm} (2)

On the other hand we know:
* $(\forall x)[|\sigma(x)| \leq |x| + k]$ \hspace{1cm} (4)
* $(4) \Rightarrow$ for each formula at any stage $|\sigma(G)| \leq p^k + k$ \hspace{1cm} (5)

$\sigma(G) \in T \subseteq 1^*$ \hspace{1cm} \Rightarrow \text{there is at most one string of each length in } T$ \hspace{1cm} (6)

$(5), (6) \Rightarrow$ we have at most $p^k + k + 1$ values for $\sigma$-mapping of a collection of formulae, \hspace{1cm} (7)

we restricted ourselves to have at most one formula corresponding to each value of $\sigma(G)$ \hspace{1cm} (8)

$(8) \Rightarrow$ we have at most $p^k + k + 1$ formulae in each collection
Proof of Theorem 1.2

* Assume $|F| = p$ \hspace{1cm} (1)
* Length of formulae will not be increased at each stage \hspace{1cm} (2)
* (1), (2) $\implies$ length of each formula at any stage $\leq p$ \hspace{1cm} (3)
Proof of Theorem 1.2

- * Assume $|F| = p$  

- Length of formulae will not be increased at each stage  

- $(1), (2) \implies$ length of each formula at any stage $\leq p$  

On the other hand we know:

- $(\forall x)[|\sigma(x)| \leq |x|^k + k]  

$(4)$
Proof of Theorem 1.2

* Assume $|F| = p$ \hspace{1cm} (1)
* Length of formulae will not be increased at each stage \hspace{1cm} (2)
* $(1), (2) \implies$ length of each formula at any stage $\leq p$ \hspace{1cm} (3)

On the other hand we know:
* $(\forall x)[|\sigma(x)| \leq |x|^k + k]$ \hspace{1cm} (4)
* $(4) \implies$ for each formula at any stage $|\sigma(G)| \leq p^k + k$ \hspace{1cm} (5)

* $(\forall x)[|\sigma(x)| \leq |x|^k + k]$ \hspace{1cm} (4)
* $(4) \implies$ for each formula at any stage $|\sigma(G)| \leq p^k + k$ \hspace{1cm} (5)
Proof of Theorem 1.2

* Assume $|F| = p$  \hspace{1cm} (1)
* Length of formulae will not be increased at each stage \hspace{1cm} (2)
* (1), (2) $\implies$ length of each formula at any stage $\leq p$ \hspace{1cm} (3)

On the other hand we know:
* $(\forall x)[|\sigma(x)| \leq |x|^k + k]$ \hspace{1cm} (4)
* (4) $\implies$ for each formula at any stage $|\sigma(G)| \leq p^k + k$ \hspace{1cm} (5)
* $\sigma(G) \in T \subseteq 1^*$ $\implies$ there is at most one string of each length in $T$ \hspace{1cm} (6)
Proof of Theorem 1.2

* Assume $|F| = p$ \hfill (1)

* Length of formulae will not be increased at each stage \( \implies \) \hfill (2)

* (1), (2) \implies \text{length of each formula at any stage} \leq p \hfill (3)

On the other hand we know:

* \((\forall x)[|\sigma(x)| \leq |x|^k + k]\) \hfill (4)

* (4) \implies \text{for each formula at any stage} \ |\sigma(G)| \leq p^k + k \hfill (5)

* \(\sigma(G) \in T \subseteq 1^* \implies \text{there is at most one string of each length in} \ T\) \hfill (6)

* (5), (6) \implies \text{we have at most} \ p^k + k + 1 \text{values for} \ \sigma\text{-mapping of a collection of formulae,} \hfill (7)
Proof of Theorem 1.2

* Assume $|F| = p$ \hspace{1cm} (1)
* Length of formulae will not be increased at each stage \hspace{1cm} (2)
* (1), (2) $\implies$ length of each formula at any stage $\leq p$ \hspace{1cm} (3)

On the other hand we know:

* $(\forall x)[|\sigma(x)| \leq |x|^k + k]$ \hspace{1cm} (4)
* (4) $\implies$ for each formula at any stage $|\sigma(G)| \leq p^k + k$ \hspace{1cm} (5)
* $\sigma(G) \in T \subseteq 1^*$ $\implies$ there is at most one string of each length in $T$ \hspace{1cm} (6)
* (5), (6) $\implies$ we have at most $p^k + k + 1$ values for $\sigma$-mapping of a collection of formulae, \hspace{1cm} (7)
* we restricted ourselves to have at most one formula corresponding to each value of $\sigma(G)$ \hspace{1cm} (8)
Proof of Theorem 1.2

* Assume $|F| = p$ (1)
* Length of formulae will not be increased at each stage (2)
* $(1), (2) \implies$ length of each formula at any stage $\leq p$ (3)

On the other hand we know:
* $(\forall x)[|\sigma(x)| \leq |x|^k + k]$ (4)
* $(4) \implies$ for each formula at any stage $|\sigma(G)| \leq p^k + k$ (5)
* $\sigma(G) \in T \subseteq 1^*$ $\implies$ there is at most one string of each length in $T$ (6)
* $(5), (6) \implies$ we have at most $p^k + k + 1$ values for $\sigma$-mapping of a collection of formulae, (7)
* we restricted ourselves to have at most one formula corresponding to each value of $\sigma(G)$ (8)
* $(8) \implies$ we have at most $p^k + k + 1$ formulae in each collection
The Pruning Technique

Proof of Theorem 1.2

- Assume $|F| = p$ \hspace{1cm} (1)
- Length of formulae will not be increased at each stage \hspace{1cm} (2)
- $(1), (2) \implies$ length of each formula at any stage $\leq p$ \hspace{1cm} (3)

On the other hand we know:
- $(\forall x)[|\sigma(x)| \leq |x|^k + k]$ \hspace{1cm} (4)
- $(4) \implies$ for each formula at any stage $|\sigma(G)| \leq p^k + k$ \hspace{1cm} (5)
- $\sigma(G) \in T \subseteq 1^*$ $\implies$ there is at most one string of each length in $T$ \hspace{1cm} (6)
- $(5), (6) \implies$ we have at most $p^k + k + 1$ values for $\sigma$-mapping of a collection of formulae, \hspace{1cm} (7)
- we restricted ourselves to have at most one formula corresponding to each value of $\sigma(G)$ \hspace{1cm} (8)
- $(8) \implies$ we have at most $p^k + k + 1$ formulae in each collection
So,

the algorithm is polynomial-time
So,

the algorithm is polynomial-time

\[ SAT \in P \implies P = NP \]
Theorem 1.4

Theorem

If there is a sparse set that is polynomial many-one hard for $\text{coNP}$, then $P = \text{NP}$.
Theorem 1.4

**Theorem**

*If there is a sparse set that is polynomial many-one hard for coNP, then P = NP.*

**Proof**

∃S and polynomial $p(n) = n^k + k$ such that

$$||\{x \in S \mid |x| \leq n\}|| \leq p(n)$$

∃σ(x) with runtime $\leq q(|x|) = |x|^j + j$ s.t. $x \in \overline{SAT} \iff \sigma(x) \in S$
Theorem

If there is a sparse set that is polynomial many-one hard for coNP, then $P = NP$.

Proof

$\exists S$ and polynomial $p(n) = n^k + k$ such that

$$\|\{x \in S \mid |x| \leq n\}\| \leq p(n)$$

$\exists \sigma(x)$ with runtime $\leq q(|x|) = |x|^j + j$ s.t. $x \in \overline{SAT} \iff \sigma(x) \in S$

Note that $p$ and $q$ are non-decreasing on $\mathbb{N}$.

We construct a polynomial-time algorithm for $SAT$, and since $SAT$ is $NP$-hard, this shows $P = NP$. 
Proof (cont.)

Similar to the other proof, we use the self-reducibility of SAT.

We proceed with an iterative algorithm that operates on a set of boolean formulae $Z$ that starts with the original formula $Z = \{F\}$. Each iteration of the algorithm makes the formulae in the new $Z$ have one less variable than the formulae in the old $Z$, so the number of iterations is linear in the number of variables of $F$, which is linear in the size of the original problem.

The next few slides describe one iteration of the algorithm.
Proof of Theorem 1.4

Proof (Iteration)
Start with an empty set $Z'$ (to become the next $Z$), and an empty set $Q$ to represent $\sigma(Z')$. 
Proof of Theorem 1.4

Proof (Iteration)
Start with an empty set $Z'$ (to become the next $Z$), and an empty set $Q$ to represent $\sigma(Z')$.

For each formula $G \in Z$, we define the two smaller formulae $G_T$ and $G_F$, where

$$G_T = G \text{ where the first variable in } G \text{ is set to True}$$

$$G_F = G \text{ where the first variable in } G \text{ is set to False}$$
Proof (Iteration, cont.)

We have $Z' = Q = \{\}$. For each formula $G \in Z$

- If $\sigma(G_T) \notin Q$, put $G_T$ in $Z'$ and $\sigma(G_T)$ in $Q$.
- If $\sigma(G_F) \notin Q$, put $G_F$ in $Z'$ and $\sigma(G_F)$ in $Q$.
- If $|Q| > p(q(|F|))$, accept.

Set $Z = Z'$. 
Proof (Iteration, cont.)

We have $Z' = Q = \{\}$.

For each formula $G \in Z$

- If $\sigma(G_T) \notin Q$, put $G_T$ in $Z'$ and $\sigma(G_T)$ in $Q$.
- If $\sigma(G_F) \notin Q$, put $G_F$ in $Z'$ and $\sigma(G_F)$ in $Q$.
- If $|Q| > p(q(|F|))$, accept.

Set $Z = Z'$.

We accept here because $Q$ can have at most $p(q(|F|))$ strings in $\overline{SAT}$ of length $\leq q(|F|)$, and we have more than that many boolean formulae whose satisfiability imply $F$'s satisfiability.

Each formula $G \in Z$ has $|G| \leq |F|$, so $|\sigma(G)| \leq q(|G|) \leq q(|F|)$, and $\overline{SAT}$ has at most $p(q(|F|))$ strings of length $\leq q(|F|)$. 

Proof (cont.)

If we have not already accepted after all iterations, we are left with a set $Z$ that contains at most $p(q(|F|))$ strings, each of which only consists of $T$ or $F$. If any of these are $T$, accept, otherwise, reject.
Validity of Proof of Theorem 1.4

Proof (cont.)

At each iteration, we transform $Z$ to $Z'$, yet we assert that some formula in $Z$ is satisfiable iff some formula in $Z'$ is satisfiable.

For each formula $G$ in $Z$, $G$ is satisfiable iff $G_T$ or $G_F$ is satisfiable. The only time we prevent adding one of these new formulae to $Z'$ is if we have another formula $H$ in $Z'$ with $\sigma(G) = \sigma(H)$, but in that case $G \in SAT \iff H \in SAT$, and we already have $H$ in $Z'$.

By induction on the number of iterations, $F$ is satisfiable iff $Z$ contains a satisfiable formula after $k$ iterations.
Proof (cont.)

Thus, if we stop during the iterations, we had too many strings for $S$, so one of them ($s$) must be in $\overline{S}$, so there is some formula $G \in Z$ with $\sigma(G) = s$, so $G \in SAT$, so $F \in SAT$.

If we don’t stop during the iterations, $F$ is satisfiable iff one of the remaining elements of $Z$ is satisfiable. There is a polynomial number of elements in $Z$, and each check takes constant time, so the total runtime is polynomial.

Thus $P = NP$. 
Theorem 1.5

Theorem

If there is a sparse coNP-complete set, then $P = NP$.

Proof

Follows immediately from Theorem 1.4.
Definition: Languages $A \subseteq \Sigma_1^*$ and $B \subseteq \Sigma_2^*$ are p-isomorphic if there is a bijection $f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that $x \in A \iff f(x) \in B$, and $f$ as well as $f^{-1}$ are polynomial-time functions.
Definition: Languages $A \subseteq \Sigma_1^*$ and $B \subseteq \Sigma_2^*$ are p-isomorphic if there is a bijection $f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that $x \in A \iff f(x) \in B$, and $f$ as well as $f^{-1}$ are polynomial-time functions.

Berman-Hartmanis conjecture: Any two NP-complete languages are p-isomorphic.
Definition: A set \( A \) is paddable if there is a one-to-one function \( f \) such that \( f \) and \( f^{-1} \) are polynomial-time functions, and \( \forall x, y, x \in A \iff f(\langle x, y \rangle) \in A \).
Definition: A set $A$ is paddable if there is a one-to-one function $f$ such that $f$ and $f^{-1}$ are polynomial-time functions, and $\forall x, y, x \in A \iff f(\langle x, y \rangle) \in A$.

All pairs of paddable NP-complete languages are $p$-isomorphic.
Definition: A set $A$ is paddable if there is a one-to-one function $f$ such that $f$ and $f^{-1}$ are polynomial-time functions, and $\forall x, y, \ x \in A \iff f(\langle x, y \rangle) \in A$.

All pairs of paddable NP-complete languages are p-isomorphic

Equivalent conjecture: all NP-complete languages are paddable
Evidence for:

- True for analogous conjecture with a restricted type of reduction ($AC^0$) [Agrawal et al. (1997)].
Evidence for:
- True for analogous conjecture with a restricted type of reduction (AC\(^0\)) [Agrawal et al. (1997)].
- True for analog in an oracle machine model [Fenner, Fortnow, & Kurtz (1992)].
Evidence against:

- k-creative sets - no p-isomorphism known [Joseph & Young (1985)]
Evidence against:

- k-creative sets - no p-isomorphism known [Joseph & Young (1985)]
- Not true for a random oracle [Kurtz, Mahaney, & Royer (1995)]
History (cont.)

- Implications of Berman-Hartmanis conjecture:
  - No NP-complete sparse languages would exist
Implications of Berman-Hartmanis conjecture:
- No NP-complete sparse languages would exist
- $P \neq NP$