The Self-Reducibility Technique

Fall 2007

Amal Fahad, Chetan Bhole,
Jonathan Gordon, Mehdi Manshadi

Department of Computer Science
University of Rochester

Part I

The Pruning Technique

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Overview

Definitions of Sparse Sets

Sparse Set

A set S is sparse if it contains at most polynomially many elements at each length.
$(\exists$ polynomial $p)$ $(\forall n)$ $[||\{x \mid x \in S \wedge |x| = n\}|| \leq p(n)]$.

- Another valid definition of sparse sets is
  $(\exists d \in \mathbb{N})$ $(\forall n)$ $[||S^{\leq n}|| \leq p_d(n)]$
  where $p_d(n) = n^d + d$

Tally Set

A set T is a tally set exactly if $T \subseteq 1^*$.

- Not all Tally sets are decidable.
  As we have seen we have T_{HP} is not decidable.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Some more definitions

Self-reducibility

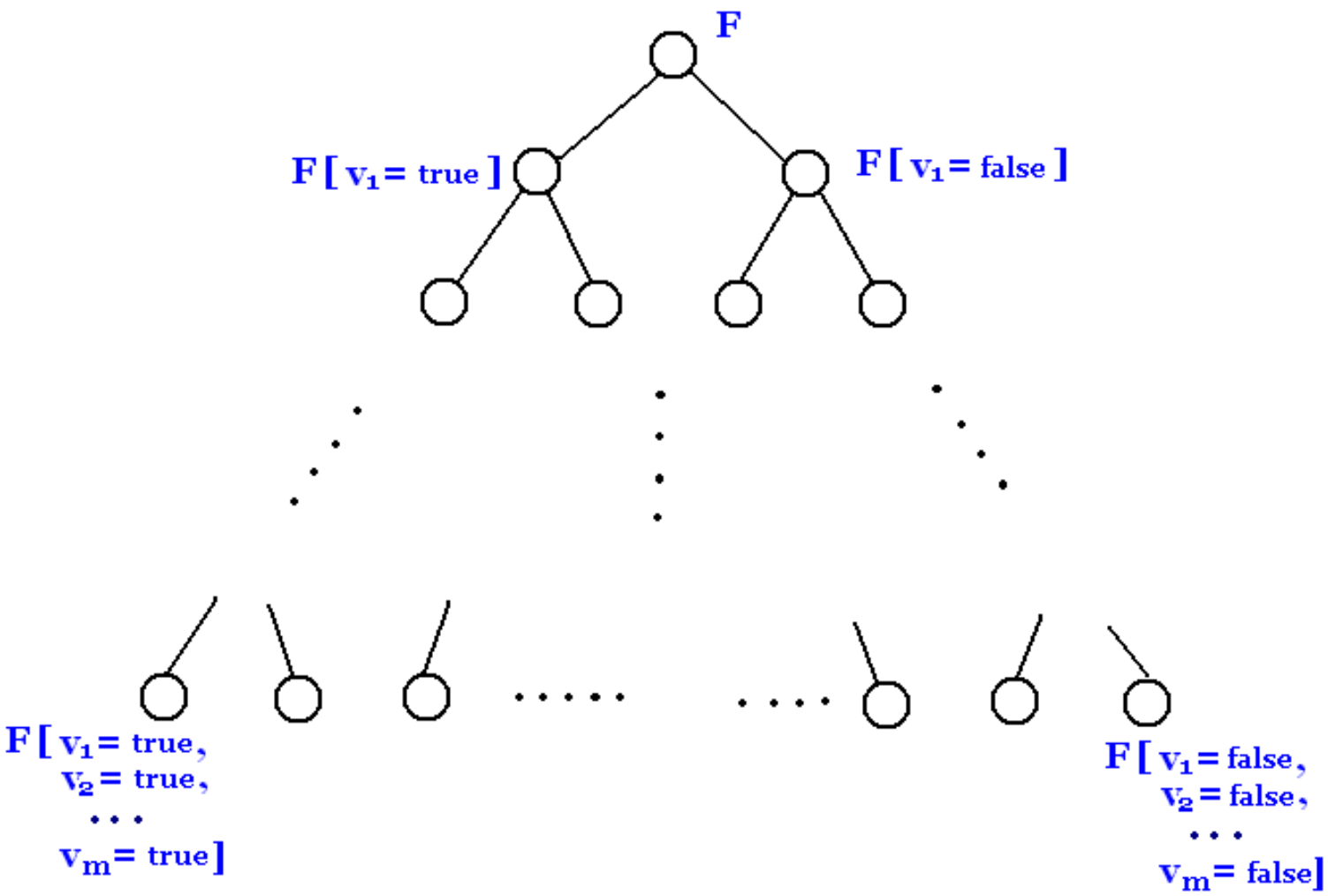
A language L is self-reducible if a deterministic poly-time oracle TM T exists such that $L = L(T^L)$ and for any input x of length n, $T^L(x)$ queries the oracle for words of length at most $n - 1$.

Disjunctive Self-Reducibility of SAT

A boolean formula having at least one variable is satisfiable if and only if either it is satisfiable with its first variable set to false or it is satisfiable with its first variable set to true.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

The disjunctive Self-Reducibility Tree



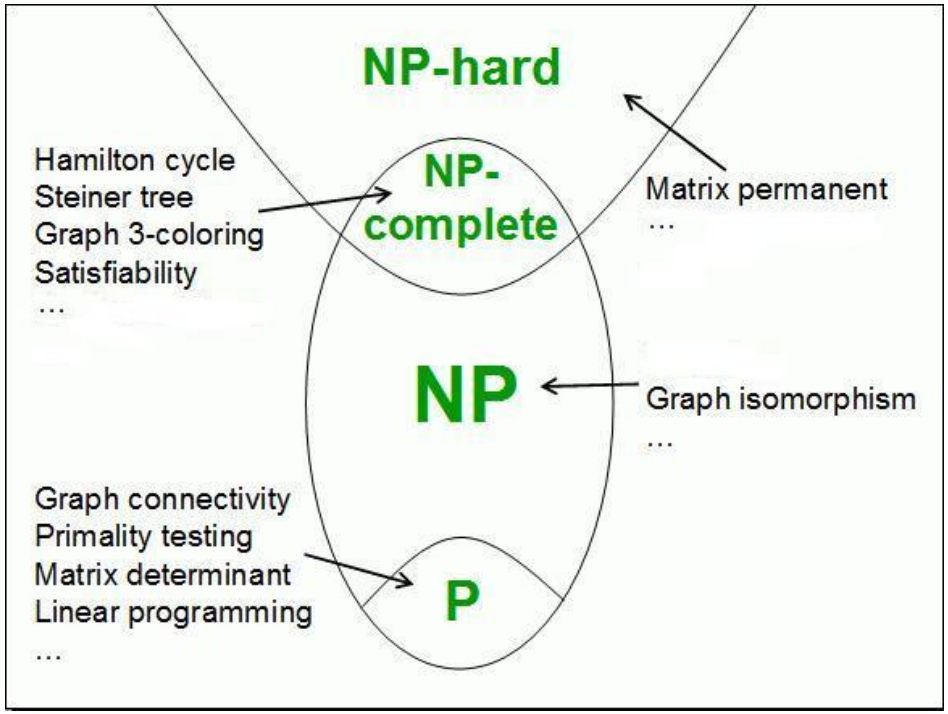
Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Some more definitions

Hardness

A set L is hard for a particular class $\mathcal{C}$ if we can reduce every set in $\mathcal{C}$ to L.
Note: L does not have to be in $\mathcal{C}$. If it is, that's completeness.

- *NP*-hard : L is *NP*-hard if $SAT \leq_m^p L$ since SAT is known to be *NP*-complete.



modified Reference:
http://www.scottaaronson.com/talks/nphard.gif

The Pruning Technique

Theorem 1.2

If there is a tally set that is NP-hard, then $P = NP$.

Corollary

If there is a tally set that is NP-complete, then $P = NP$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Proof

Given: $\exists T$ s.t. $T \subseteq 1^*$ and $SAT \leq_m^p T$, hence,

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

where g is a deterministic, polynomial-time function reducing SAT to T and k is an integer, which must exist since g is computable by some deterministic polynomial-time Turing machine that outputs at most one character per step.

- We create a deterministic, polynomial-time algorithm for SAT using the tree-pruning technique. And so if we can decide SAT in polynomial time, then $P = NP$.
- Let there be v_1 to v_m variables in our formula.
- Our algorithm has stages 0, 1, ..., $m + 1$ described below.

Algorithm

Stage 0

Outputs $\mathcal{C} = \{F\}$ where F is the original formula

Stage i

Input to stage i: $\mathcal{C} = \{F_1, \ldots, F_l\}$ (the output from the previous stage)
Step 1: Replace v_i by True or False to get

$$\begin{aligned}\mathcal{C} \quad = \quad & \{F_1[v_i = \text{True}], F_2[v_i = \text{True}], \ldots F_l[v_i = \text{True}], \\ & F_1[v_i = \text{False}], F_2[v_i = \text{False}], \ldots, F_l[v_i = \text{False}]\}\end{aligned}$$

Step 2: $\mathcal{C}' = \emptyset$
Step 3: For each f in $\mathcal{C}$ do:

1. Compute $g(f)$
2. If $g(f) \in 1^*$ and for no formula $h \in \mathcal{C}'$ does $g(f) = g(h)$, then add f to $\mathcal{C}'$.

Output of stage i: $\mathcal{C} = \mathcal{C}'$

Stage m+1 : The Last stage.

Input is $\mathcal{C}$ which is now a variable-free formula collection.
F is satisfiable if an element in $\mathcal{C}$ is true.

Correctness

1. We know $SAT \leq_m^p T$, hence

$$F \in SAT \iff g(F) \in T$$

if $g(F) \notin T$, F cannot be in SAT. Hence in step 3, part 2 we do not include such F_i in our new collection.

2.

$$F_1 \in SAT \iff g(F_1) \in T$$

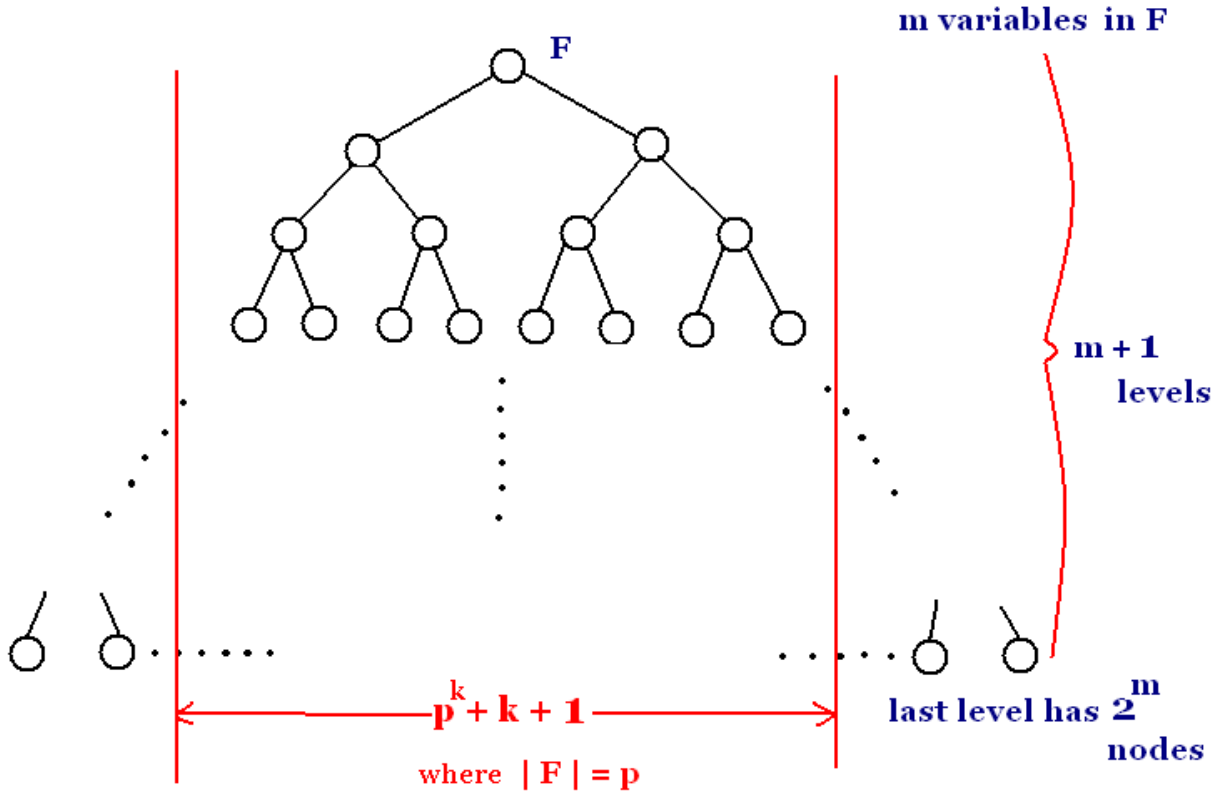
$$F_2 \in SAT \iff g(F_2) \in T$$

But if $g(F_1) = g(F_2)$ then

$$F_1 \in SAT \iff F_2 \in SAT$$

Meaning it is sufficient to show that either one of $F_1 \in SAT$ or $F_2 \in SAT$ and so in step 3, part 2, we need only include one such formula in our new collection.

Proof of polynomial time



- Number of formulae at the output of each stage $\leq p^k + k + 1$ where $|F| = p$
- $m + 1$ levels where m is the number of variables in our formula
- At each node we call $g(F_i)$ which is also bounded by $p^k + k$

Hence a polynomial-time algorithm ($O(mp^{2k})$) for SAT, so $P = NP$.

The Pruning Technique

Theorem 1.4

If there is a sparse set that is $\leq_m^p$-hard for *coNP*, then $P = NP$.

Corollary 1.5

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

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Proof

Given:

1. $\exists$ S s.t. S is sparse i.e.
   $(\forall n)$ $[||S^{\leq n}|| \leq p_d(n)]$ where $p_d(n) = n^d + d$
2. $\overline{SAT} \leq_m^p S$.
   where g is a deterministic, polynomial-time function reducing $\overline{SAT}$ to S and let k be such that

$$(\forall x)|g(x)| \leq |x|^k + k$$

- Like before we create a deterministic, polynomial-time algorithm that allows us to decide whether the formula F is Satisfiable or not.
- Let there be v_1 to v_m variables in our formula.
- Our algorithm has stages 0, 1, ..., $m+1$ described below. We can terminate before stage $m+1$.

Stage 0

Outputs $\mathcal{C} = \{F\}$ where F is the original formula

Algorithm

Stage i

Input to stage i: $\mathcal{C} = \{F_1, \ldots, F_l\}$ (the output from the previous stage)

Step 1: Let

$$\begin{aligned} \mathcal{C} \quad = \quad & \{F_1[v_i = \text{True}], \ldots F_l[v_i = \text{True}], \\ & F_1[v_i = \text{False}], \ldots F_l[v_i = \text{False}]\} \end{aligned}$$

Step 2: Set $\mathcal{C}' = \emptyset$

Step 3: For each formula f in $\mathcal{C}$:

1. Compute $g(f)$
2. If for no formula $h \in \mathcal{C}'$ does $g(f) = g(h)$, add f to $\mathcal{C}'$

Step 4: If $\mathcal{C}'$ contains at least $p_d(p_k(|F|)) + 1$ elements, stop and immediately declare that $F \in SAT$.

Stage $m + 1$

If some member of $\mathcal{C}$ evaluates to true, $F \in SAT$. Otherwise, $F \notin SAT$.

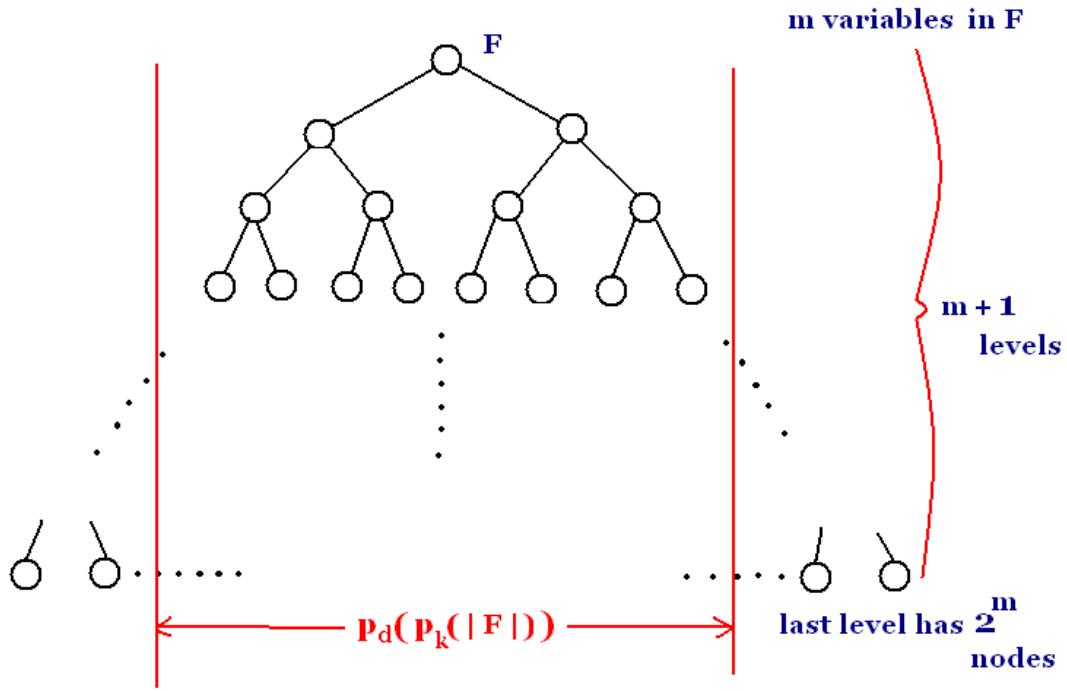
Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Correctness

1. If we reach stage $m+1$ then trivially we have a collection of formulae in which all variables are assigned and we can check if F is in SAT by checking if any one of the formulae in our collection is satisfied.
2. If we stop abruptly at step 4 of any stage then we use the following argument to show that $F \in SAT$
   We know,
   $$F_i \in \overline{SAT} \iff g(F_i) \in S$$
   We had $p_d(p_k(|F|))+1$ or more unique elements generated by g. But our sparse set can only be as big as $p_d(p_k(|F|))$. That means at least one element computed by g was not in S.
   But,
   $$g(F_i) \notin S \Rightarrow F_i \notin \overline{SAT}$$
   Therefore, $F_i \in SAT$ meaning $F \in SAT$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Proof of polynomial time



- Number of formulae at the output of each stage is $\leq p_d(p_k(|F|))$
- At the most $m+1$ levels where m is the number of variables in F
- At each node we call $g(F_i)$ which is also bounded by $|F|^k + k$

The algorithm clearly runs in polynomial time on the size of the input F and we can show whether F is in SAT or not thus proving $P = NP$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Summary

We showed that we can show $P = NP$ using the pruning method on self-reducibility trees in the following cases:

1. If there is a tally set that is $\leq_m^p$-hard for NP, then $P = NP$
2. If there is a sparse set that is $\leq_m^p$-hard for $coNP$, then $P = NP$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

References

- Lane Hemaspaandra and Mitsunori Ogihara. *The Complexity Theorem Companion*. Springer: Springer, 2002.

Part II

Mahaney's Theorem

Overview

Introduction

Last Lecture

We proved:

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

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Introduction

Last Lecture

We proved:

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

This Lecture

We will prove Mahaney's Theorem:

- If there is an *NP*-complete sparse set, then $P = NP$.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Proof of Theorem 1.4

Theorem 1.4

If there is a sparse set that is $\leq_m^p$-hard for *coNP*, then $P = NP$

Suppose S is a sparse set that is $\leq_m^p$-hard for *coNP*, then $\overline{SAT} \leq_m^p S$. There is a polynomial-time function g:

$$g : \overline{SAT} \rightarrow S$$

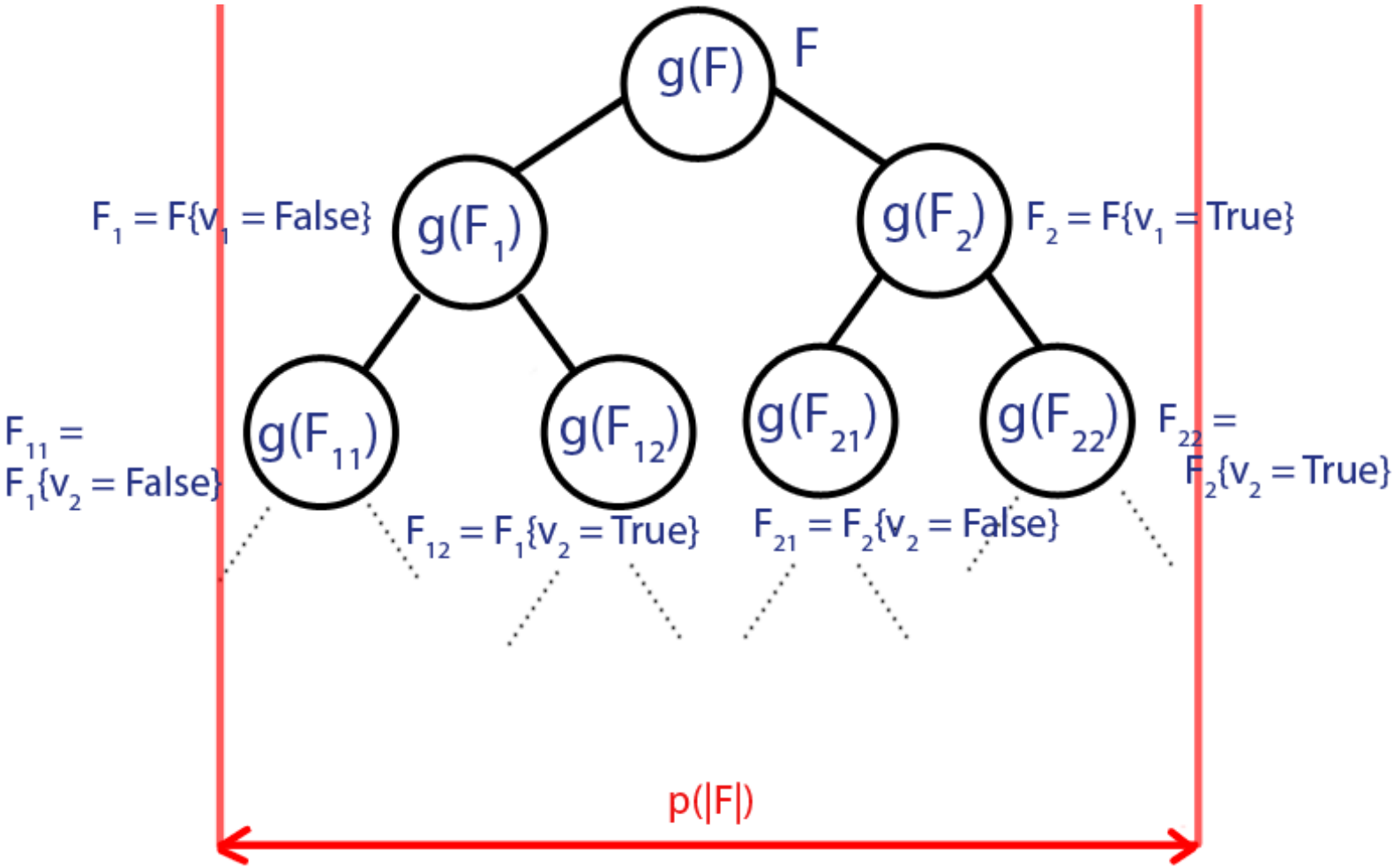
$$\forall x \ |g(x)| \leq P_g(|x|)$$

$$\forall n \ C_s(n) \leq P_s(n)$$

for monotonically increasing polynomials P_g and P_s and where $C_s(n) = ||S^{\leq n}||$ is called the census function.

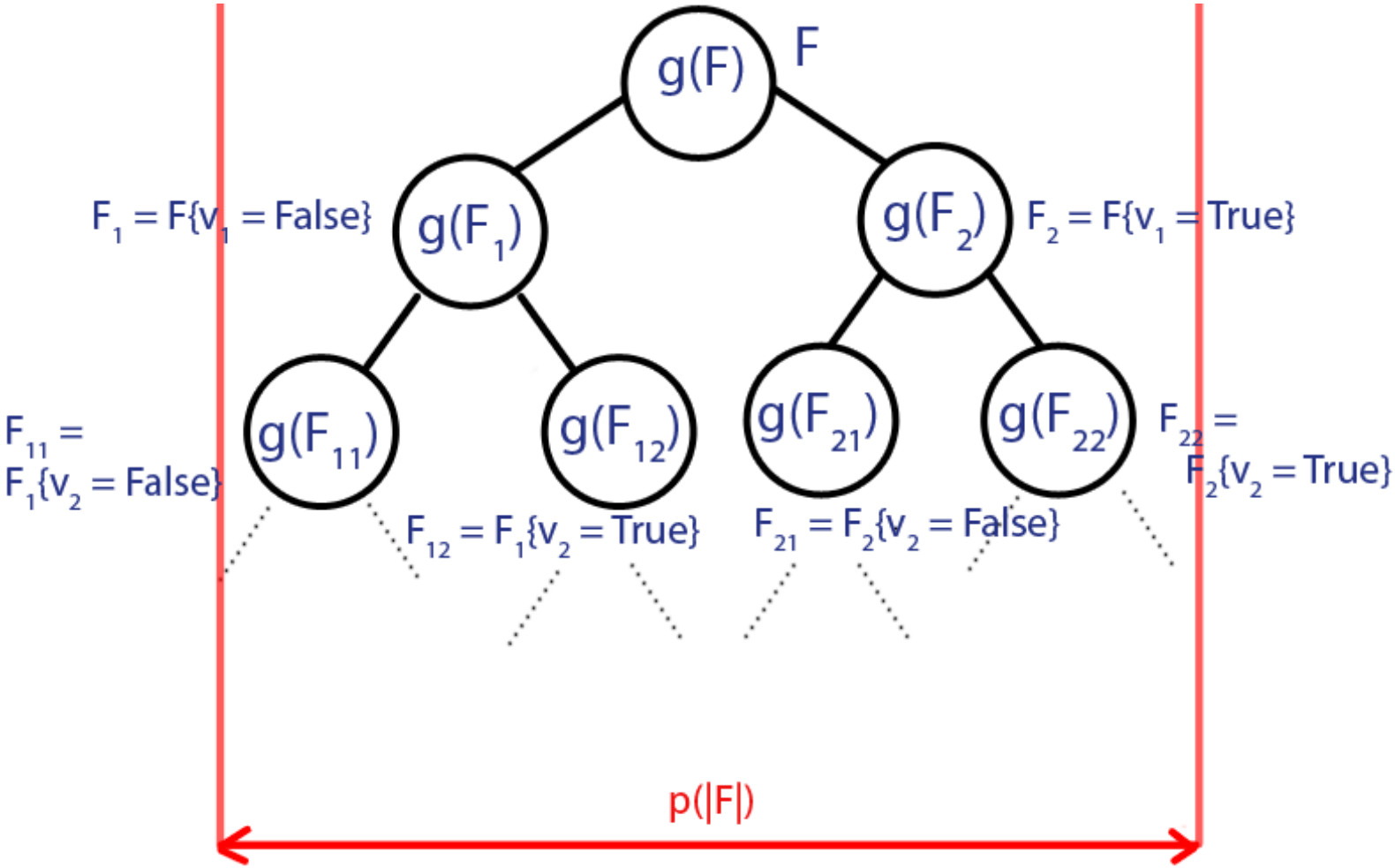
Breadth-first Search Method

$$g : \overline{SAT} \rightarrow S$$
$$\forall x \; |g(x)| \leq P_g(|x|)$$
$$\forall n \; C_s(n) \leq P_s(n)$$



Breadth-first Search Method

$$g : \overline{SAT} \rightarrow S$$
$$\forall x \ |g(x)| \leq P_g(|x|)$$
$$\forall n \ C_s(n) \leq P_s(n)$$



This doesn't work if S is NP-complete. Why? $SAT \leq_m^p S$

Depth-first Search Method

Another proof for Theorem 1.4 (a depth-first search method):

Depth-first Search Method

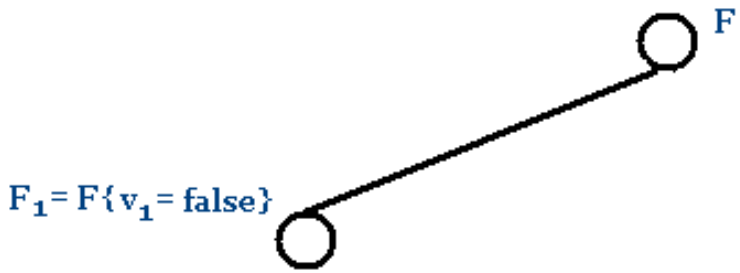
Another proof for Theorem 1.4 (a depth-first search method):

SL = { g(false) }

Depth-first Search Method

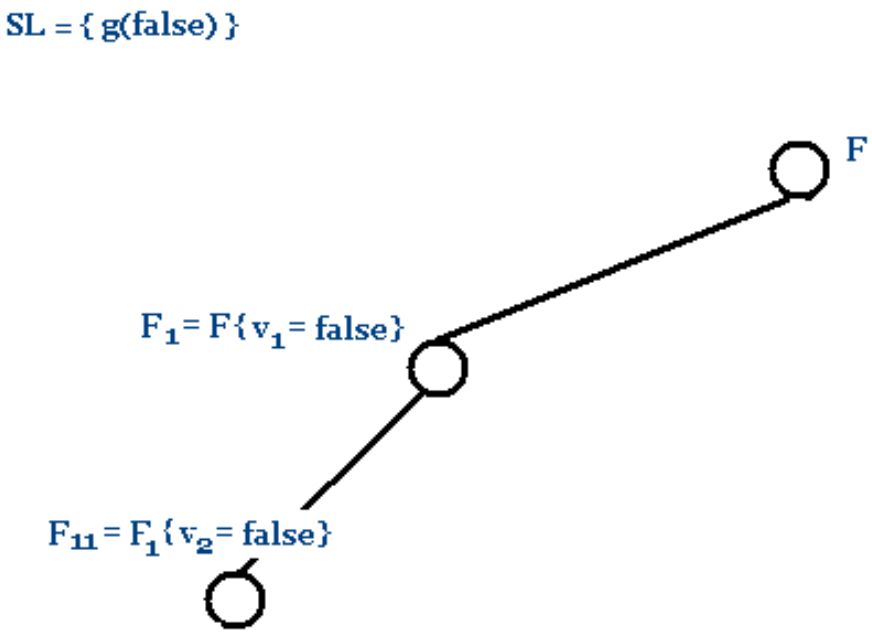
Another proof for Theorem 1.4 (a depth-first search method):

SL = { g(false) }



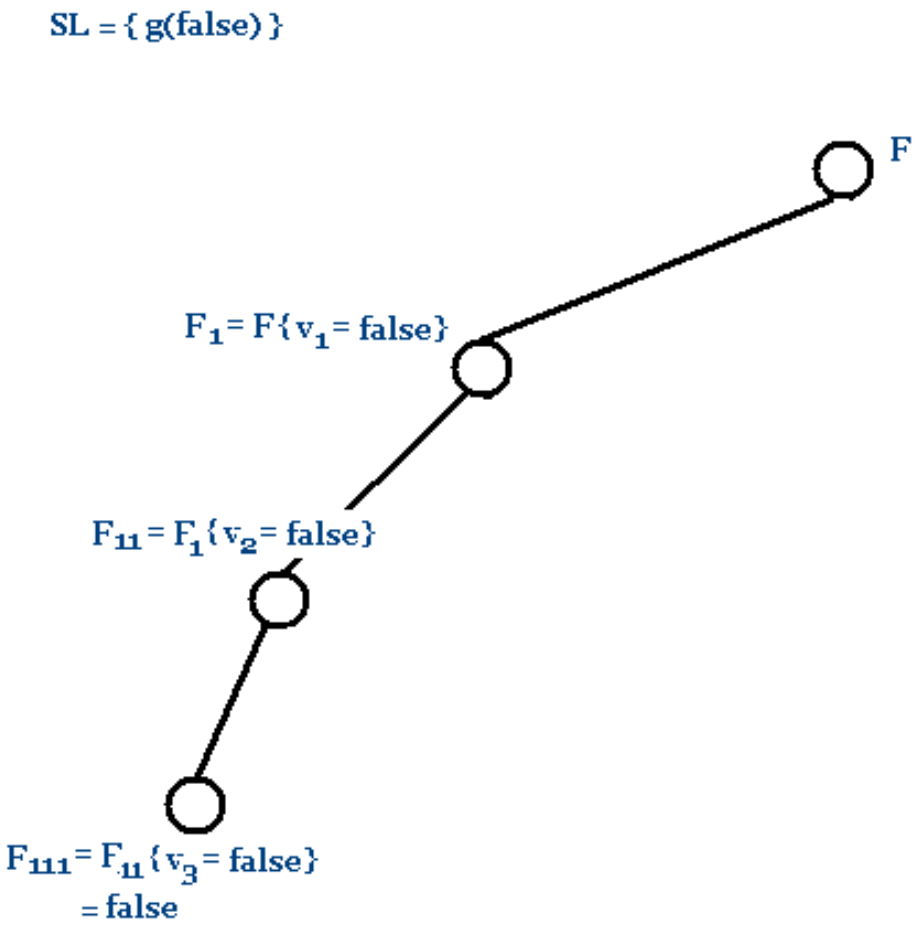
Depth-first Search Method

Another proof for Theorem 1.4 (a depth-first search method):



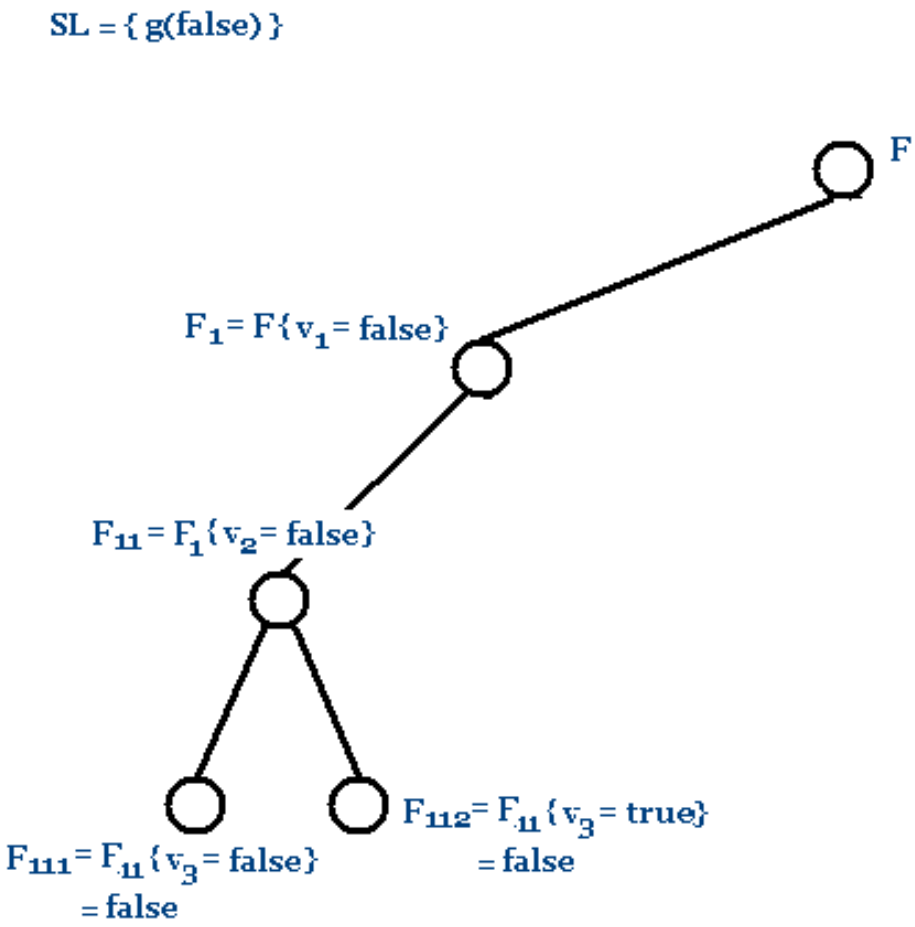
Depth-first Search Method

Another proof for Theorem 1.4 (a depth-first search method):



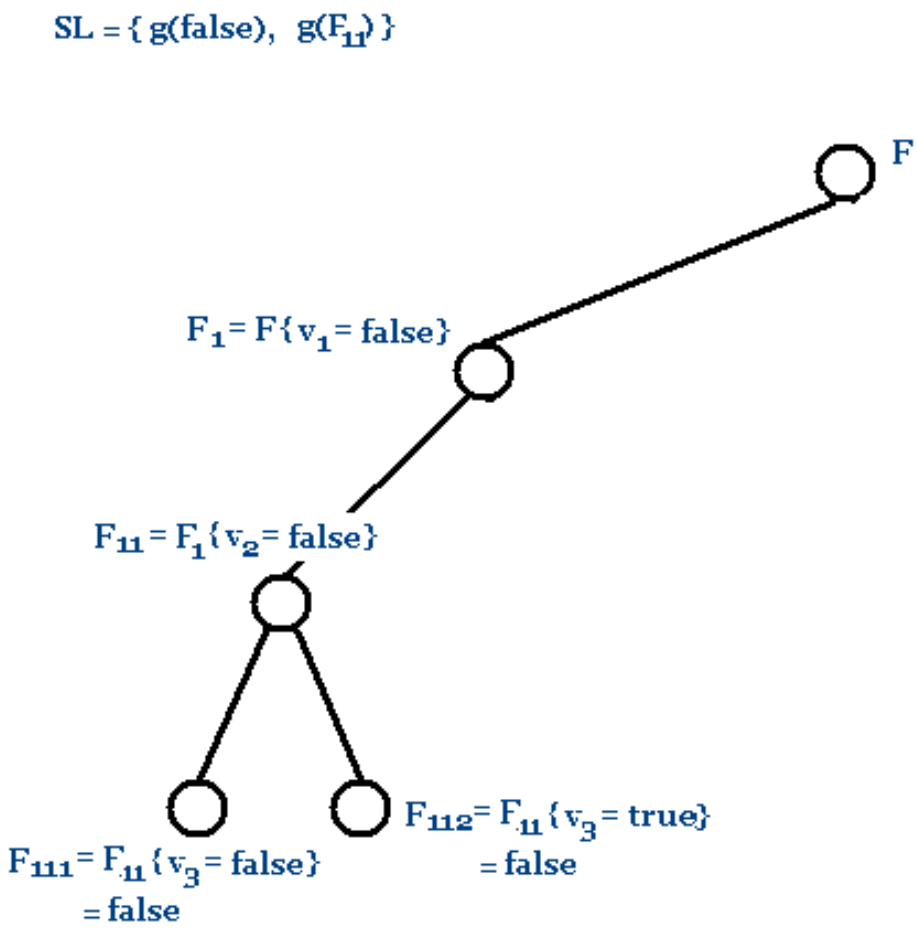
Depth-first Search Method

Another proof for Theorem 1.4 (a depth-first search method):



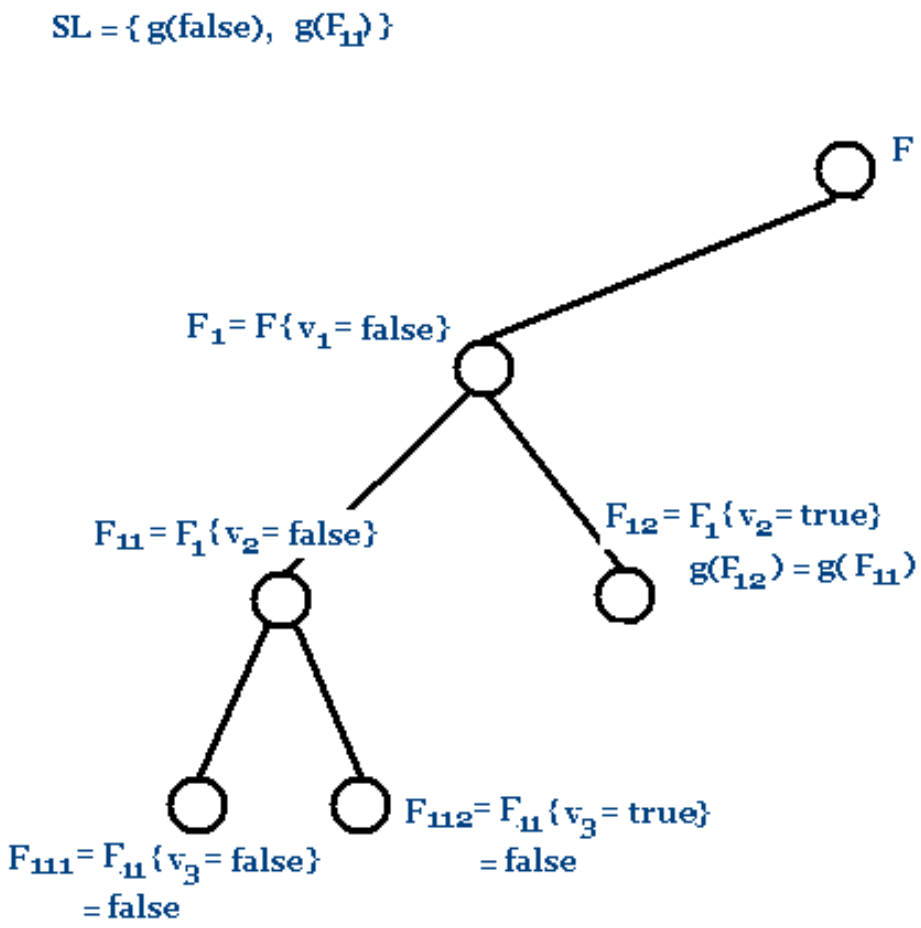
Depth-first Search Method

Another proof for Theorem 1.4 (a depth-first search method):



Depth-first Search Method

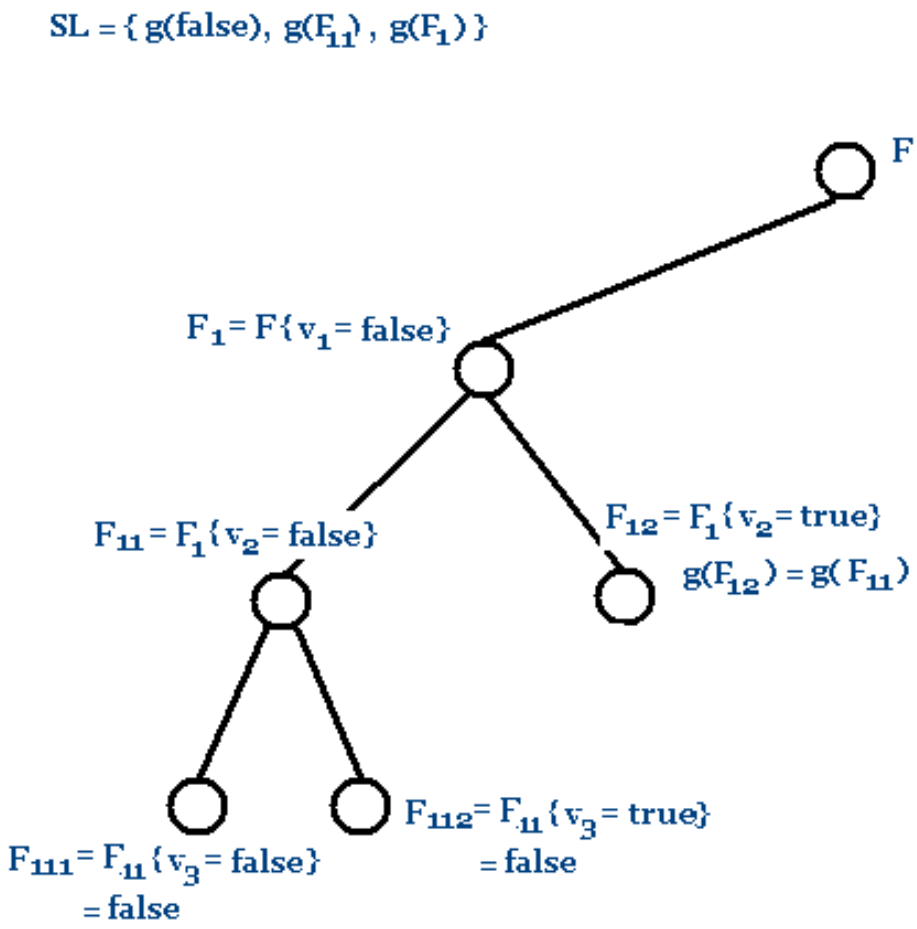
Another proof for Theorem 1.4 (a depth-first search method):



Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

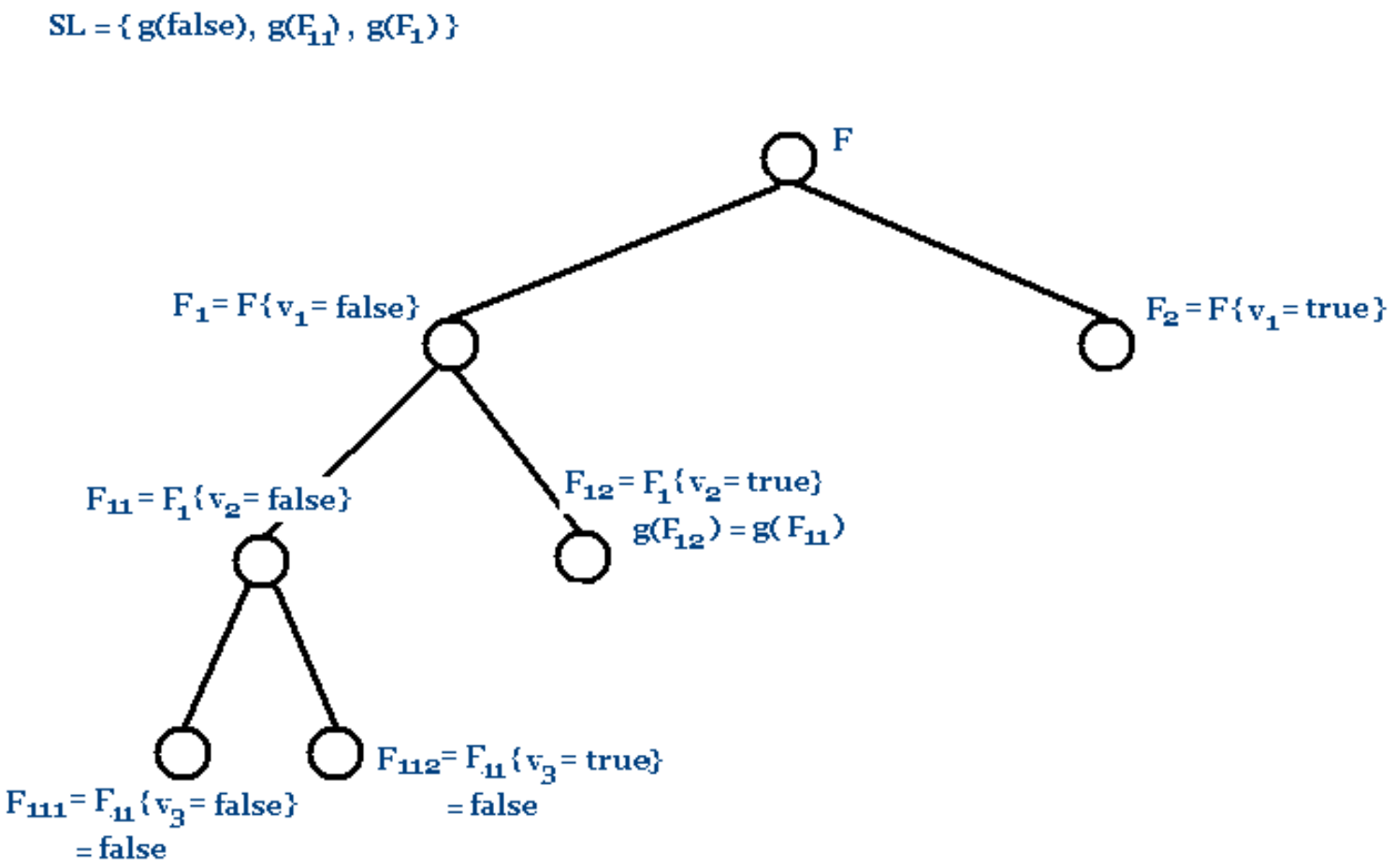
Depth-first Search Method

Another proof for Theorem 1.4 (a depth-first search method):



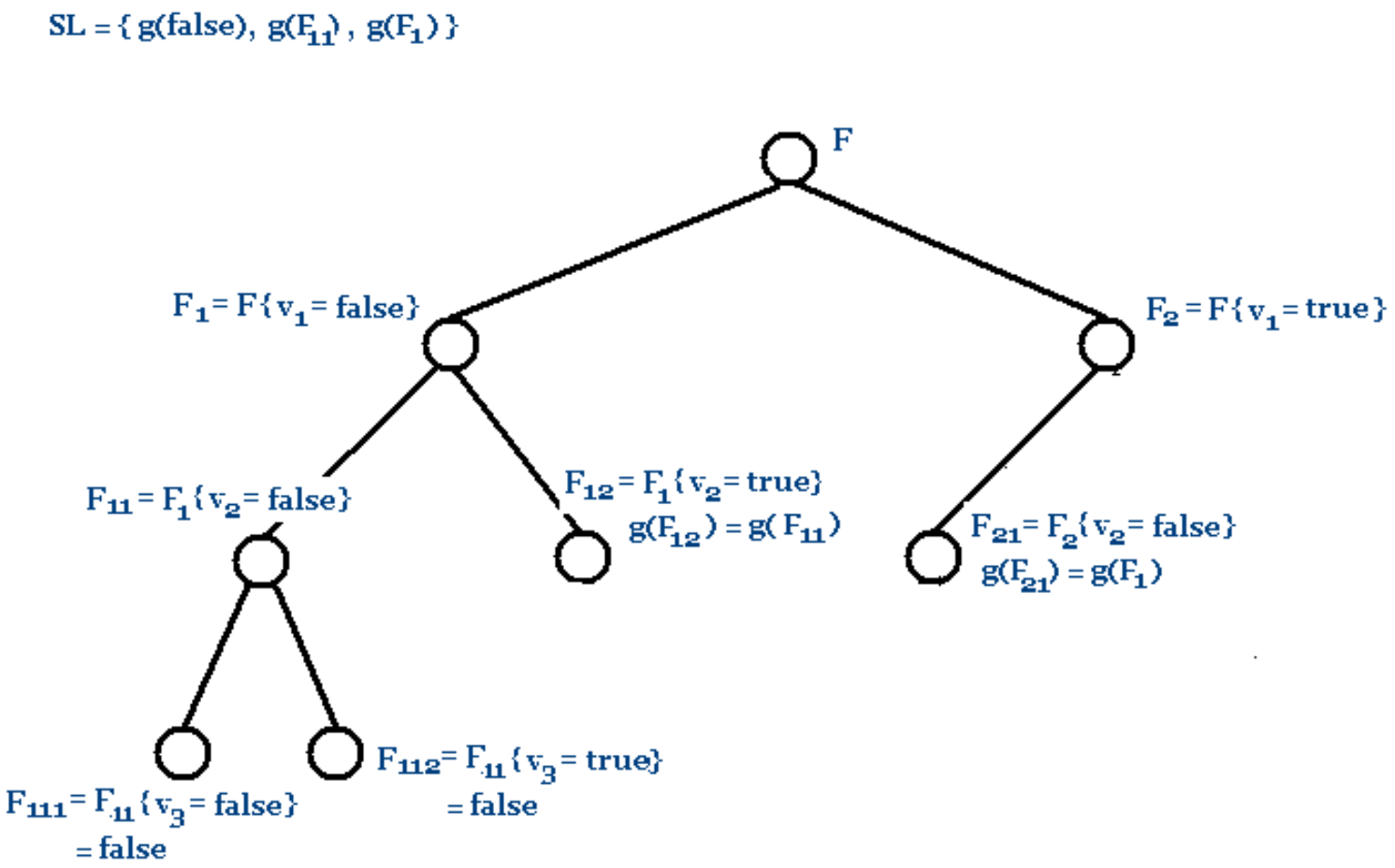
Depth-first Search Method

Another proof for Theorem 1.4 (a depth-first search method):



Depth-first Search Method

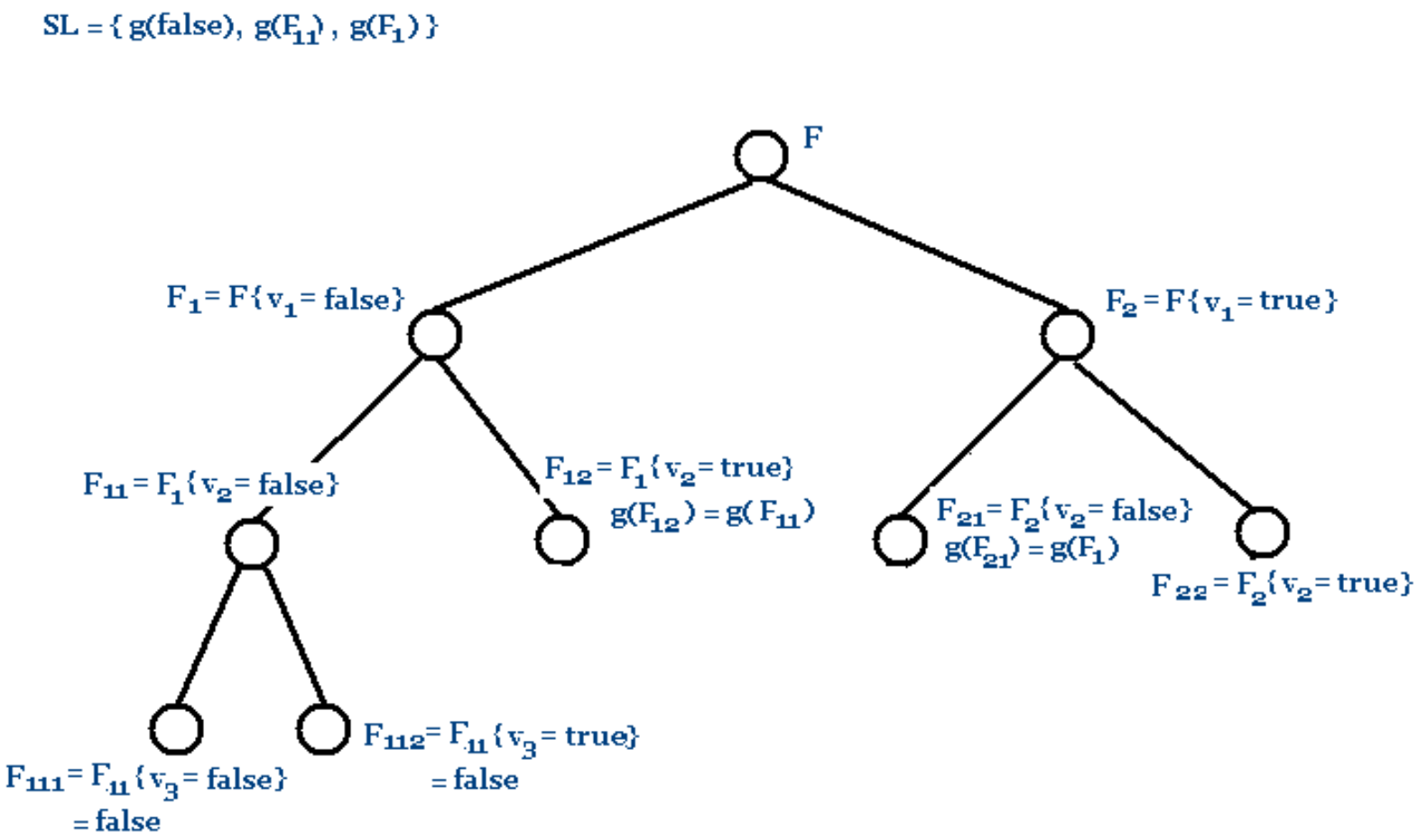
Another proof for Theorem 1.4 (a depth-first search method):



Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Depth-first Search Method

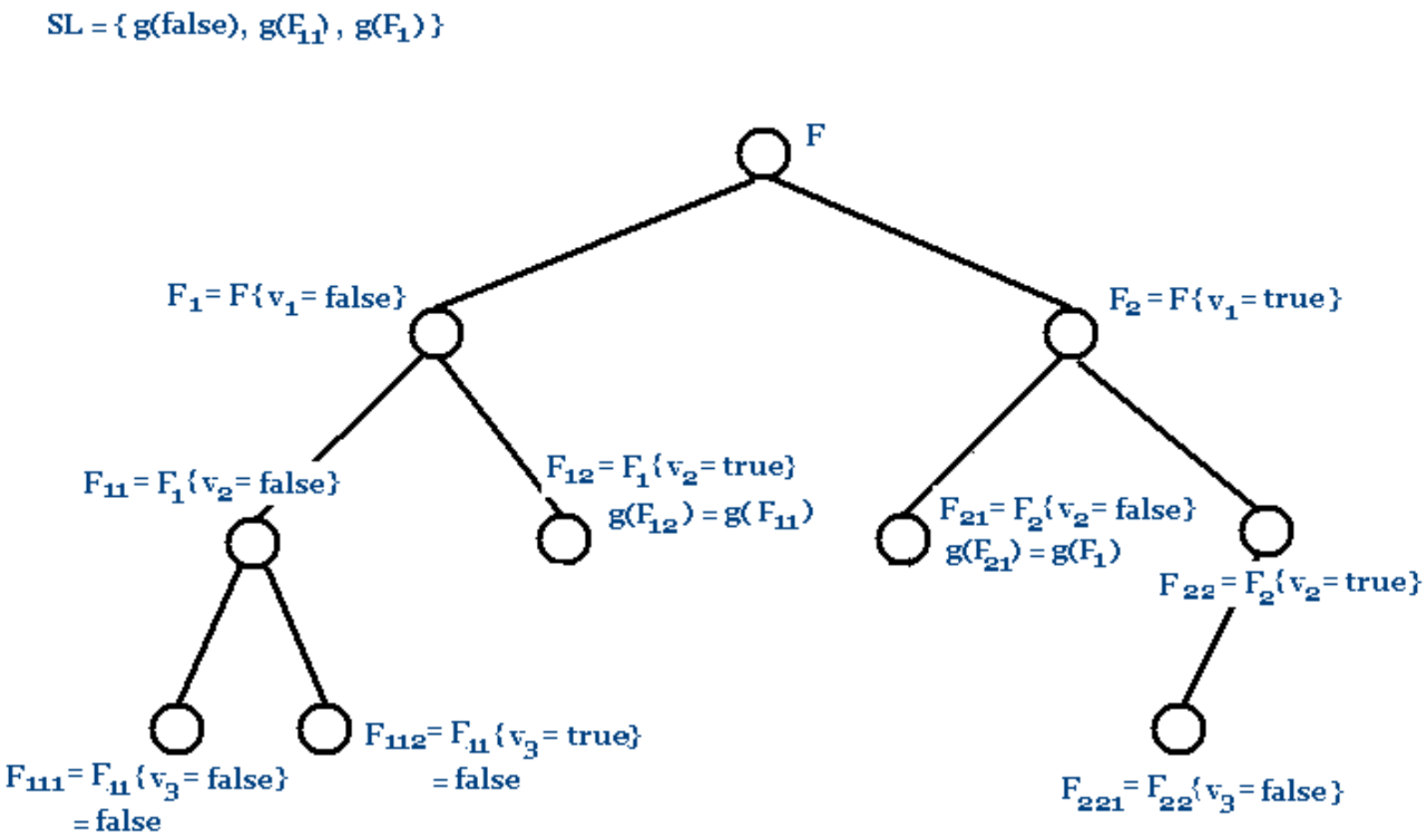
Another proof for Theorem 1.4 (a depth-first search method):



Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Depth-first Search Method

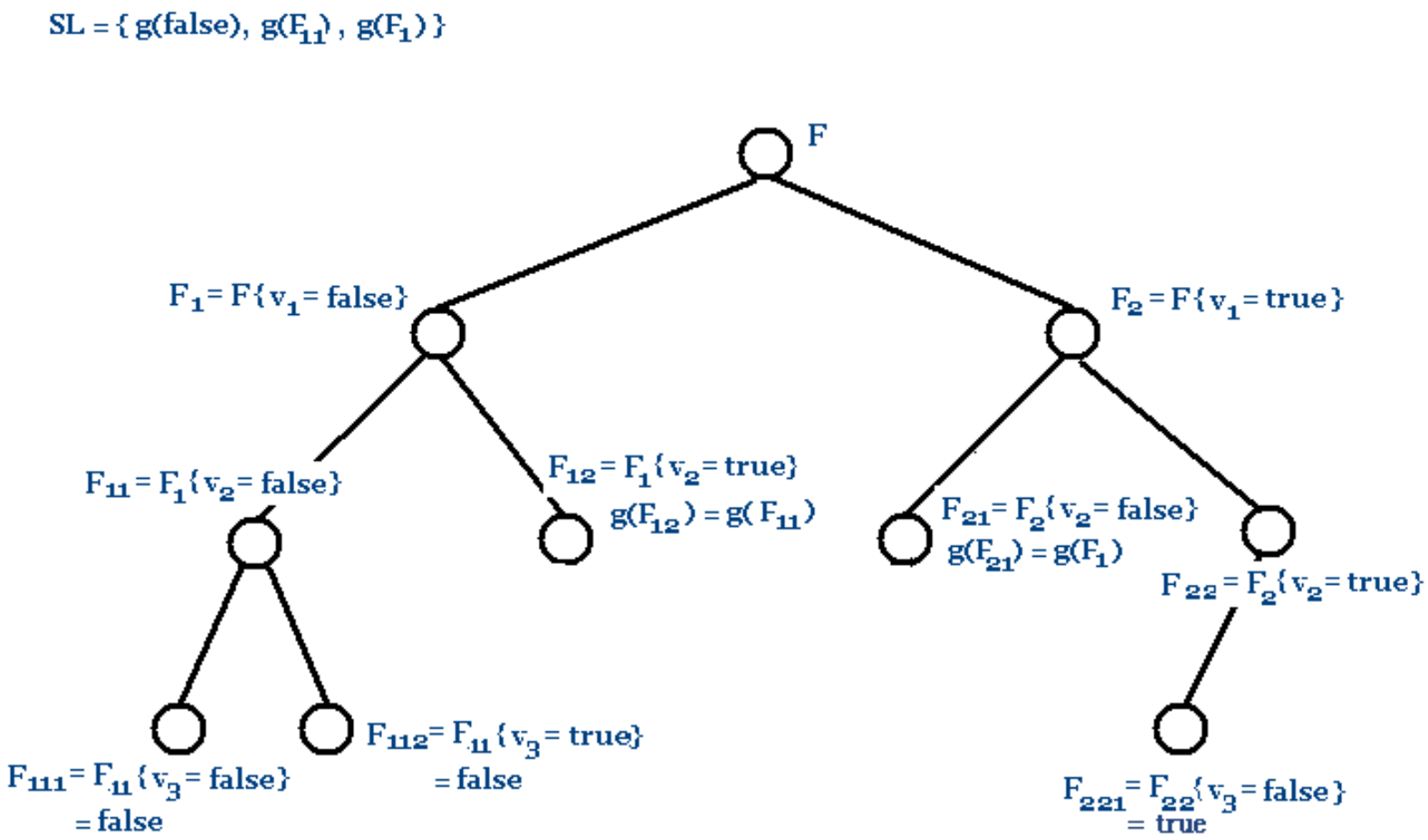
Another proof for Theorem 1.4 (a depth-first search method):



Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Depth-first Search Method

Another proof for Theorem 1.4 (a depth-first search method):



Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Depth-first Algorithm

Decide(F)

SL = { g(False) };
Search(F);
Declare unsatisfiable and halt.
End.

Depth-first Algorithm

Decide(F)

```
    SL = { g(False) };
    Search(F);
    Declare unsatisfiable and halt.
End.
```

Search(G)

```
    if G = True
        Declare satisfiable and halt.
    else if g(G) is in SL
        return;
    else
        For v, the first variable in G:
        Search(G{v = False});
        Search(G{v = True});
        Add g(G) to SL;
        return;
End.
```


Correctness

Correctness

Argument

- The only way that Search declares a formula satisfiable is by finding a satisfying assignment.

Correctness

Argument

- The only way that Search declares a formula satisfiable is by finding a satisfying assignment.
- That is: If Search declares G is satisfiable, then G is definitely satisfiable.

Correctness

Argument

- The only way that Search declares a formula satisfiable is by finding a satisfying assignment.
- That is: If Search declares G is satisfiable, then G is definitely satisfiable.
- The only way that we don't expand a node is that we already know that the corresponding formula is unsatisfiable.

Correctness

Argument

- The only way that Search declares a formula satisfiable is by finding a satisfying assignment.
- That is: If Search declares G is satisfiable, then G is definitely satisfiable.
- The only way that we don't expand a node is that we already know that the corresponding formula is unsatisfiable.
- That is: If G is satisfiable, then finally Search finds a satisfying assignment.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Running Time

How many nodes does the Search method visit?

Running Time

How many nodes does the Search method visit?

Unsatisfiable Nodes

$$|g(F_i)| \leq P_g(|F_i|) \leq P_g(|F|)$$
$$||SL|| \leq P_s(P_g(|F|))$$

Running Time

How many nodes does the Search method visit?

Unsatisfiable Nodes

$$|g(F_i)| \leq P_g(|F_i|) \leq P_g(|F|)$$
$$||SL|| \leq P_s(P_g(|F|))$$

Claim: No two interior nodes can be labelled with the same value in SL unless they are on the same path from the root.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

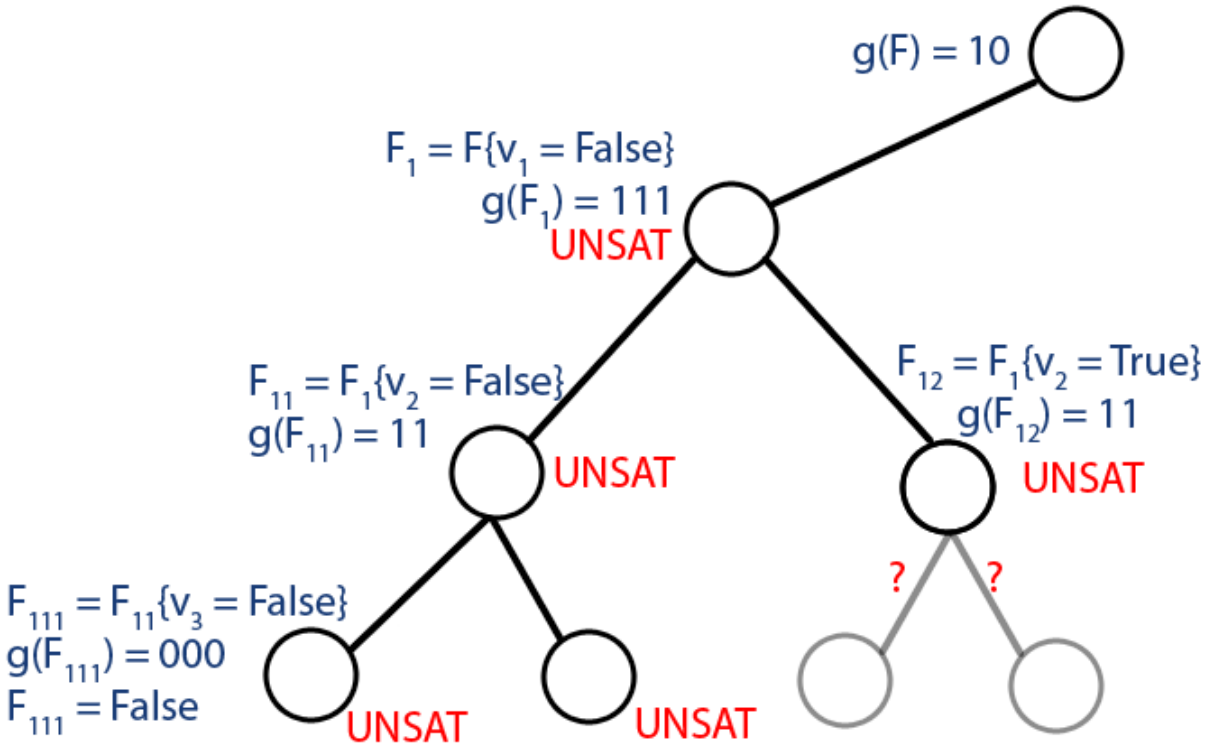
Running Time

How many nodes does the Search method visit?

Unsatisfiable Nodes

$$|g(F_i)| \leq P_g(|F_i|) \leq P_g(|F|)$$
$$||SL|| \leq P_s(P_g(|F|))$$

Claim: No two interior nodes can be labelled with the same value in SL unless they are on the same path from the root.



Maximum number of unsatisfiable interior nodes: $m \cdot P_s(P_g(|F|))$ where m is the number of variables in F.

Running Time

How many nodes does the Search method visit?

Satisfiable Interior Nodes

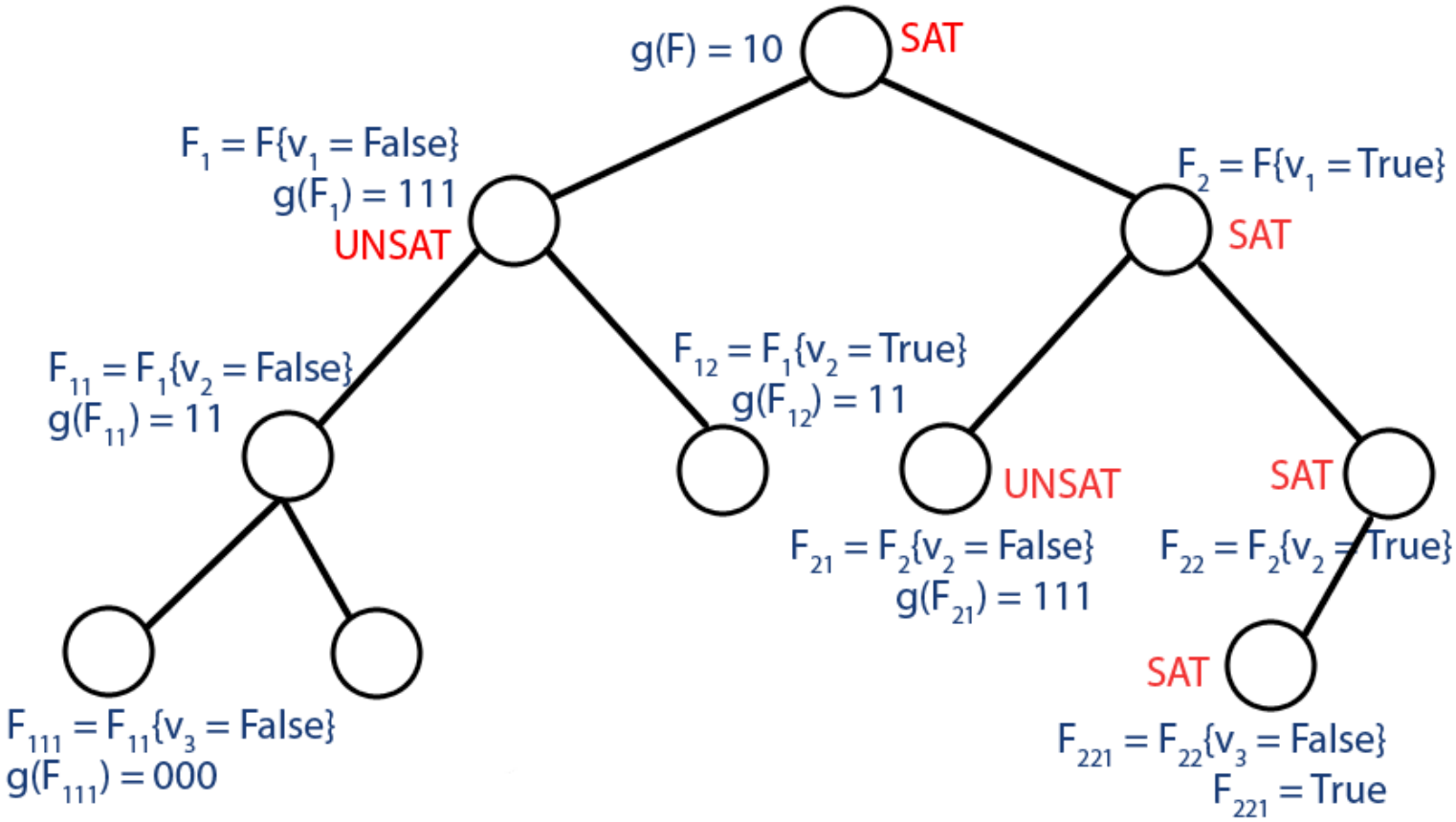
Maximum: m

Running Time

How many nodes does the Search method visit?

Satisfiable Interior Nodes

Maximum: m

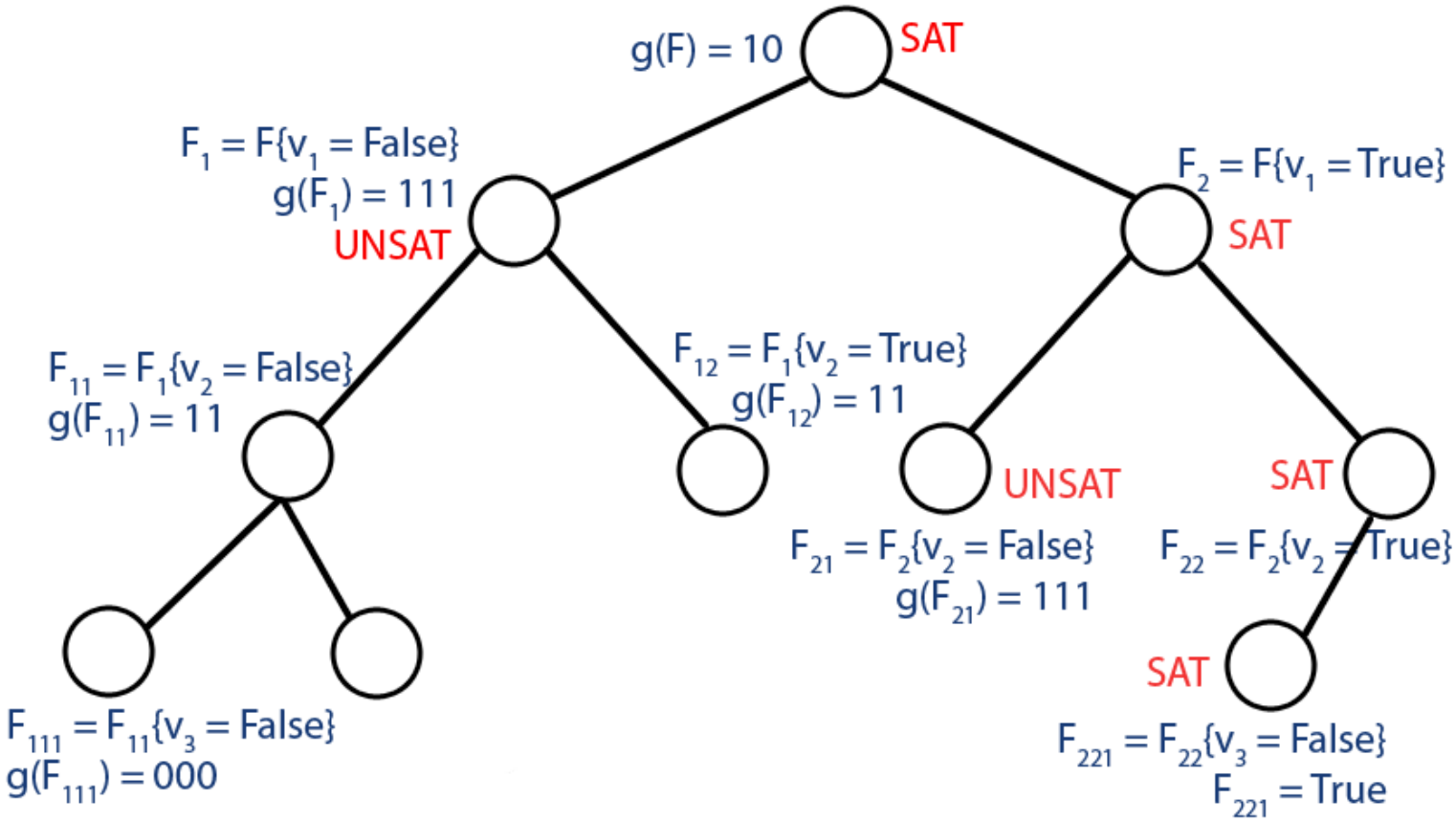


Running Time

How many nodes does the Search method visit?

Satisfiable Interior Nodes

Maximum: m



Maximum number of interior nodes that the Search method visits: $m + m \cdot P_s(P_g(|F|))$

Mahaney's Theorem

Theorem

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

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Mahaney's Theorem

Theorem

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

Suppose S is an NP-complete sparse set, then there is a polynomial-time function f:

$$f : SAT \rightarrow S$$

$$f : \overline{SAT} \rightarrow \overline{S}$$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Mahaney's Theorem

Theorem

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

Suppose S is an NP-complete sparse set, then there is a polynomial-time function f:

$$f : SAT \rightarrow S$$

$$f : \overline{SAT} \rightarrow \overline{S}$$

Is $\overline{S}$ in NP? If yes,

$$h : \overline{S} \rightarrow S$$

$$g = h \circ f : \overline{SAT} \rightarrow S$$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Is $\overline{S}$ in NP?

A nondeterministic, polynomial-time TM that accepts $\overline{S}$:

On input x,

Let $n = |x|$

Let $k = C_s(n)$

In a nondeterministic fashion, guess distinct strings $s_1, s_2, \ldots, s_k$ (such that $|s_i| \leq n$)

If all of these strings are in S,

then accept x if it is not among the s_i's.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Is $\overline{S}$ in NP?

A nondeterministic, polynomial-time TM that accepts $\overline{S}$:

On input x,
    Let $n = |x|$
    Let $k = C_s(n)$?
    In a nondeterministic fashion, guess distinct strings $s_1, s_2, \ldots, s_k$ (such that $|s_i| \leq n$)
    If all of these strings are in S,
        then accept x if it is not among the s_i's.

But: This algorithm is wrong. $C_s(n)$ may not be P-time computable.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Pseudocomplement of S

A nondeterministic, polynomial-time TM that accepts $\overline{S}^p$ (the pseudo-complement of S):

On input $\langle x, k, 0^n \rangle$

- If $|x| > n$, reject.
- If $k > P_s(n)$, reject.
- Guess distinct strings $s_1, s_2, \ldots, s_k$ in a nondeterministic fashion (such that $|s_i| \leq n$) and guess proofs that each belongs to S.
- If this guess succeeded, then accept x if x is not among the s_i's.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Pseudocomplement of S

A nondeterministic, polynomial-time TM that accepts $\overline{S}^p$ (the pseudo-complement of S):

On input $\langle x, k, 0^n \rangle$

- If $|x| > n$, reject.
- If $k > P_s(n)$, reject.
- Guess distinct strings $s_1, s_2, \ldots, s_k$ in a nondeterministic fashion (such that $|s_i| \leq n$) and guess proofs that each belongs to S.
- If this guess succeeded, then accept x if x is not among the s_i's.

$k = C_s(n)$: $\langle x, k, 0^n \rangle \in \overline{S}^p \Leftrightarrow x \in \overline{S}$ (for $|x| \leq n$)

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Pseudocomplement of S

A nondeterministic, polynomial-time TM that accepts $\overline{S}^p$ (the pseudo-complement of S):

On input $\langle x, k, 0^n \rangle$

- If $|x| > n$, reject.
- If $k > P_s(n)$, reject.
- Guess distinct strings $s_1, s_2, \ldots, s_k$ in a nondeterministic fashion (such that $|s_i| \leq n$) and guess proofs that each belongs to S.
- If this guess succeeded, then accept x if x is not among the s_i's.

$k = C_s(n)$: $\langle x, k, 0^n \rangle \in \overline{S}^p \Leftrightarrow x \in \overline{S}$ (for $|x| \leq n$)

(What happens if $k > C_s(n)$ or $k < C_s(n)$?)

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Pruning Functions

$$f : \overline{SAT} \rightarrow \overline{S} \quad \forall x \; |f(x)| \leq P_f(|x|)$$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Pruning Functions

$$f : \overline{SAT} \rightarrow \overline{S} \quad \forall x \ |f(x)| \leq P_f(|x|)$$
$$h : \overline{S}^p \rightarrow S \quad \forall x \ |h(x)| \leq P_h(|x|)$$

where P_f and P_h are monotonically increasing polynomials.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Pruning Functions

$$f : \overline{SAT} \to \overline{S} \quad \forall x \; |f(x)| \leq P_f(|x|)$$
$$h : \overline{S}^p \to S \quad \forall x \; |h(x)| \leq P_h(|x|)$$

where P_f and P_h are monotonically increasing polynomials.

Let $n = P_f(|F|)$

Pruning functions: $g_k(F_i) = h(\langle f(F_i), k, 0^n \rangle)$ (for $k \leq P_s(n)$)

$$\begin{aligned} |g_k(F_i)| &\leq P_h(|\langle f(F_i), k, 0^n \rangle|) \\ &\leq P_g(|F|) \end{aligned}$$

for some polynomial P_g.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Pruning Functions

$$f : \overline{SAT} \to \overline{S} \quad \forall x \; |f(x)| \leq P_f(|x|)$$
$$h : \overline{S}^p \to S \quad \forall x \; |h(x)| \leq P_h(|x|)$$

where P_f and P_h are monotonically increasing polynomials.

Let $n = P_f(|F|)$

Pruning functions: $g_k(F_i) = h(\langle f(F_i), k, 0^n \rangle)$ (for $k \leq P_s(n)$)

$$\begin{aligned} |g_k(F_i)| &\leq P_h(|\langle f(F_i), k, 0^n \rangle|) \\ &\leq P_g(|F|) \end{aligned}$$

for some polynomial P_g.

When $k = C_s(n)$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Pruning Functions

$$f : \overline{SAT} \rightarrow \overline{S} \quad \forall x \ |f(x)| \leq P_f(|x|)$$
$$h : \overline{S}^p \rightarrow S \quad \forall x \ |h(x)| \leq P_h(|x|)$$

where P_f and P_h are monotonically increasing polynomials.

Let $n = P_f(|F|)$

Pruning functions: $g_k(F_i) = h(\langle f(F_i), k, 0^n \rangle)$ (for $k \leq P_s(n)$)

$$\begin{aligned} |g_k(F_i)| &\leq P_h(|\langle f(F_i), k, 0^n \rangle|) \\ &\leq P_g(|F|) \end{aligned}$$

for some polynomial P_g.

When $k = C_s(n)$

$F_i \in \overline{SAT} \Leftrightarrow g_k(F_i) \in S$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Pruning Functions

$$f : \overline{SAT} \rightarrow \overline{S} \quad \forall x \ |f(x)| \leq P_f(|x|)$$
$$h : \overline{S}^p \rightarrow S \quad \forall x \ |h(x)| \leq P_h(|x|)$$

where P_f and P_h are monotonically increasing polynomials.

Let $n = P_f(|F|)$

Pruning functions: $g_k(F_i) = h(\langle f(F_i), k, 0^n \rangle)$ (for $k \leq P_s(n)$)

$$\begin{aligned} |g_k(F_i)| &\leq P_h(|\langle f(F_i), k, 0^n \rangle|) \\ &\leq P_g(|F|) \end{aligned}$$

for some polynomial P_g.

When $k = C_s(n)$

$F_i \in \overline{SAT} \Leftrightarrow g_k(F_i) \in S$

$g_k : \overline{SAT} \rightarrow S$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Pruning Functions

$$f : \overline{SAT} \rightarrow \overline{S} \quad \forall x \ |f(x)| \leq P_f(|x|)$$
$$h : \overline{S}^p \rightarrow S \quad \forall x \ |h(x)| \leq P_h(|x|)$$

where P_f and P_h are monotonically increasing polynomials.

Let $n = P_f(|F|)$

Pruning functions: $g_k(F_i) = h(\langle f(F_i), k, 0^n \rangle)$ (for $k \leq P_s(n)$)

$$\begin{aligned} |g_k(F_i)| &\leq P_h(|\langle f(F_i), k, 0^n \rangle|) \\ &\leq P_g(|F|) \end{aligned}$$

for some polynomial P_g.

When $k = C_s(n)$

$F_i \in \overline{SAT} \Leftrightarrow g_k(F_i) \in S$

$g_k : \overline{SAT} \rightarrow S$

$||S_L|| \leq P_s(P_g(|F|))$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Polynomial Algorithm for *SAT*

Decide(F)

for $k = 0$ to $P_s(n)$ {
  $SL = \{g_k(False)\}$
  Run the Search method on F with pruning function g_k.
  If the number of interior nodes visited by the Search method exceeds $m + m \cdot P_s(P_g(|F|))$, halt the search for this k.
}
Declare unsatisfiable and halt.

End.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Polynomial Algorithm for *SAT*

Decide(F)

for $k = 0$ to $P_s(n)$ {
  $SL = \{g_k(False)\}$
  Run the Search method on F with pruning function g_k.
  If the number of interior nodes visited by the Search method exceeds $m + m \cdot P_s(P_g(|F|))$, halt the search for this k.
}
Declare unsatisfiable and halt.

End.

Running time: polynomial in $|F|$.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Polynomial Algorithm for *SAT*

Decide(F)

for $k = 0$ to $P_s(n)$ {
  $SL = \{g_k(False)\}$
  Run the Search method on F with pruning function g_k.
  If the number of interior nodes visited by the Search method exceeds $m + m \cdot P_s(P_g(|F|))$, halt the search for this k.
}
Declare unsatisfiable and halt.
End.

Running time: polynomial in $|F|$.

Correctness

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Polynomial Algorithm for *SAT*

Decide(F)

for $k = 0$ to $P_s(n)$ {
  $SL = \{g_k(False)\}$
  Run the Search method on F with pruning function g_k.
  If the number of interior nodes visited by the Search method exceeds $m + m \cdot P_s(P_g(|F|))$, halt the search for this k.
}
Declare unsatisfiable and halt.
End.

Running time: polynomial in $|F|$.

Correctness

- Decide declares F as satisfiable only if it finds a satisfying assignment.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Polynomial Algorithm for *SAT*

Decide(F)

for $k = 0$ to $P_s(n)$ {
  $SL = \{g_k(False)\}$
  Run the Search method on F with pruning function g_k.
  If the number of interior nodes visited by the Search method exceeds $m + m \cdot P_s(P_g(|F|))$, halt the search for this k.
}
Declare unsatisfiable and halt.
End.

Running time: polynomial in $|F|$.

Correctness

- Decide declares F as satisfiable only if it finds a satisfying assignment.
- If F is satisfiable, when $k = C_s(n)$, the Search method finally will find a satisfying assignment.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Polynomial Algorithm for *SAT*

Decide(F)

for $k = 0$ to $P_s(n)$ {
  $SL = \{g_k(False)\}$
  Run the Search method on F with pruning function g_k.
  If the number of interior nodes visited by the Search method exceeds $m + m \cdot P_s(P_g(|F|))$, halt the search for this k.
}
Declare unsatisfiable and halt.
End.

Running time: polynomial in $|F|$.

Correctness

- Decide declares F as satisfiable only if it finds a satisfying assignment.
- If F is satisfiable, when $k = C_s(n)$, the Search method finally will find a satisfying assignment.
- F is declared as satisfiable if and only if F is satisfiable.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Summary

Mahaney's Theorem

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

What about an NP-hard sparse set?

Summary

Mahaney's Theorem

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

What about an NP-hard sparse set?

Theorem 1.9 (see Hemaspaandra-Ogihara)

NP has sparse $\leq_m^p$-hard sets if and only if NP has sparse $\leq_m^p$-complete sets.

Any questions?

References

- S. Mahaney. "Sparse Sets and Reducibilities". *Studies in Complexity Theory*, pages 63-118. John Wiley and Sons, 1986.
- Lane Hemaspaandra and Mitsunori Ogihara. *The Complexity Theorem Companion*. Springer: Springer, 2002.

Part III

The Hartmanis–Immerman–Sewelson Encoding

Overview

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Introduction

The Question

Is there any sparse set in $NP - P$?
That is, is there a sparse set in NP that's so hard it has no polynomial-time algorithm?

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Introduction

The Question

Is there any sparse set in $NP - P$?
That is, is there a sparse set in NP that's so hard it has no polynomial-time algorithm?

Such a set would, necessarily, not be NP-complete. Why?

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Introduction

The Question

Is there any sparse set in $NP - P$?
That is, is there a sparse set in NP that's so hard it has no polynomial-time algorithm?

Such a set would, necessarily, not be NP-complete. Why?

Remember Mahaney's Theorem from last lecture: We proved that if any sparse set is NP-complete, then $P = NP$. Well, if $P = NP$, then $NP - P$ is empty, so our set can't exist.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

E and NE

E and NE are exponential-time analogs of P and NP. Recall:

$$P = \bigcup_k DTIME[n^k]$$

$$NP = \bigcup_k NTIME[n^k]$$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

E and NE

E and NE are exponential-time analogs of P and NP. Recall:

$$P = \bigcup_k DTIME[n^k]$$

$$NP = \bigcup_k NTIME[n^k]$$

$$E = \bigcup_{c>0} DTIME[2^{cn}]$$

$$NE = \bigcup_{c>0} NTIME[2^{cn}]$$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

E and NE

E and NE are exponential-time analogs of P and NP. Recall:

$$P = \bigcup_k DTIME[n^k]$$
$$NP = \bigcup_k NTIME[n^k]$$

$$E = \bigcup_{c>0} DTIME[2^{cn}]$$
$$NE = \bigcup_{c>0} NTIME[2^{cn}]$$

And just as $P \subseteq NP$, $E \subseteq NE$.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Theorem

Theorem

The following are equivalent:

1. $E = NE$
2. $NP - P$ *contains no sparse sets*
3. $NP - P$ *contains no tally sets*

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Theorem Proof: Part One

Lemma 1.19

If $NP - P$ contains no tally sets, then $E = NE$.

Proof: Since we know that $E \subseteq NE$, we can prove this by showing that $NE \subseteq E$.

Set up

$L \in NE$, so there must exist a nondeterministic, exponential-time TM N s.t. $L(N) = L$.

Define a tally set

$L' = \{1^k \mid (\exists x \in L)[k = (1x)_2]\}$

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Proof $L' \in NP$

Does $L' \in NP$? We give an NPTM for L':

Algorithm for L'

On input y,

- Reject if y is not of the form 1^k (for some $k > 0$).
- Otherwise, simulate nondeterministically $N(w)$ where w is all digits of k except the leftmost 1.

Run-time: $|w|$ is logarithmic in the length of y, so the run-time is at most $O(2^{c \log |y|}) = O(|y|^c)$ for some c. So, the algorithm runs nondeterministically (because N is nondeterministic) and runs in polynomial-time.

Since L' is a tally set in NP and our assumption is that $NP - P$ contains no tally sets, this means that $L' \in P$.

Proof $L \in E$

There is a deterministic, polynomial-time TM M s.t. $L(M) = L'$. We use this to construct a TM that accepts L:

Algorithm for L

On input y,

- Compute $b = 1^{(1y)_2}$
- Simulate $M(b)$.
- Accept iff M accepts.

Run-time: Since M is polynomial-time and $|b| \leq 2^{|y|+1}$, the number of steps $M(b)$ requires is exponential-time: $(2^{|y|+1})^c = 2^{c|y|+c}$.

Thus $L \in E$. Since our proof holds for any $L \in NE$, $NE \subseteq E$ and thus our lemma is proved:

So, if $NP - P$ contains no tally sets, then $E = NE$.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Warm-up Proof

We want to prove:

Lemma 1.21

If $E = NE$ then $NP - P$ contains no sparse sets

First, though, we prove the simpler claim:

Prove

If $E = NE$ then $NP - P$ contains no tally sets.

$E = NE \Rightarrow NP - P$ contains no tally sets

Let L be a tally set in NP.

Define

$L' = \{x \mid (x \text{ is } 0 \text{ or } x \text{ is a binary string of nonzero length with no leading zeros}) \text{ and } 1^{(x)_2} \in L\}$

Using the NPTM N that accepts L, we can give an algorithm to accept L' in nondeterministic exponential time:

Algorithm for L'

On input y,

- Reject if ($y \neq 0$ and has leading zeros) or $y = \epsilon$.
- Otherwise, simulate $N(x)$ nondeterministically where $x = 1^{(y)_2}$.

$L' \in NE$, so (by our assumption) $L' \in E$.

$L \in P$

Since $L' \in E$, there is a deterministic, exponential-time TM that accepts L'. We'll call this TM ME. We can use it to construct a polynomial-time algorithm for L.

Algorithm for L

On input y,

- Reject if $y \notin 1^k$ for some k
- Otherwise, write k as 0 if $k = 0$ or as $(k)_2$ with no leading zeros.
- Then simulate ME for L' on $(k)_2$.

Why is this polynomial-time?

Why doesn't this work for sparse sets?

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

$L \in P$

Since $L' \in E$, there is a deterministic, exponential-time TM that accepts L'. We'll call this TM ME. We can use it to construct a polynomial-time algorithm for L.

Algorithm for L

On input y,

- Reject if $y \notin 1^k$ for some k
- Otherwise, write k as 0 if $k = 0$ or as $(k)_2$ with no leading zeros.
- Then simulate ME for L' on $(k)_2$.

Why is this polynomial-time?

Why doesn't this work for sparse sets?
This proof uses the fact that the length of a string in a tally set determines the string (a string of length n must be 1^n). This is not true for all sparse sets. We need a new encoding.

Theorem Proof: Part Two

Lemma 1.21

If $E = NE$ then $NP - P$ contains no sparse sets.

Let L be a sparse set in NP. We need to show it's in P.

There is some polynomial q such that $(\forall n)[||L^{=n}|| \leq q(n)]$

Hartmanis–Immerman–Sewelson Encoding

$$\begin{aligned} L' = & \{0\#n\#k \mid ||L^{=n}|| \geq k\} \cup \\ & \{1\#n\#c\#i\#j \mid (\exists z_1, z_2, \ldots, z_c \in L^{=n}) \\ & [z_1 <_{\text{lex}} z_2 <_{\text{lex}} \cdots <_{\text{lex}} z_c \wedge \text{ the } j\text{th bit of } z_i \text{ is } 1]\} \end{aligned}$$

Since $L \in NP$, we can show $L' \in NE$.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

$L' \in NE$

Algorithm for L'

On input y,

- If first bit is 0
  - Get binary values of n and k
  - Guess k distinct strings that are n bits long.
  - If there is such a set of strings that each string is accepted by N_L then accept.
  - Otherwise, reject.
- If first bit is 1
  - Get binary values of n, c, i, and j.
  - Guess c distinct strings that are n bits long.
  - Put them in lexical order.
  - If there is a set such that each string is accepted by N_L and the jth bit of the ith string is 1, accept.
  - Otherwise, reject.

Algorithm is nondeterministic, exponential-time. $L' \in NE$, so $L' \in E$ (by our assumption). This means there is a deterministic, exponential-time TM M' which accepts L'.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

$L \in P$

We give an algorithm to prove that $L \in P$:

On input x,

- Let $n = |x|$
- Simulate M' on $0\#n\#0$, $0\#n\#1$, $\ldots$, $0\#n\#q(n)$ to see which of them belong to L'
- Let $c = \max\{k \mid 0 \leq k \leq q(n) \wedge 0\#n\#k \in L' = ||L^{=n}||\}$
- Now simulate M' on
  $1\#n\#c\#1\#1, 1\#n\#c\#1\#2, \ldots, 1\#n\#c\#1\#n$,
  $1\#n\#c\#2\#1, 1\#n\#c\#2\#2, \ldots, 1\#n\#c\#2\#n$,
  $\ldots$
  $1\#n\#c\#c\#1, 1\#n\#c\#c\#2, \ldots, 1\#n\#c\#c\#n$
- If x is in the set of the n-length strings given by M''s answers, then accept. Otherwise, reject.

This is polynomially many queries. $L' \in E$, but each of the polynomially-many queries to the TM M' is of length $O(\log n)$. Thus the algorithm is polynomial-time.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Summary

Theorem

The following are equivalent:

1. $E = NE$
2. $NP - P$ *contains no sparse sets*
3. $NP - P$ *contains no tally sets*

We have shown that if $NP - P$ has no tally sets, $E = NE$ and if $E = NE$, $NP - P$ has no sparse sets, thus we have proved our theorem.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

References

- Lane Hemaspaandra and Mitsunori Ogihara. *The Complexity Theorem Companion*. Springer: Springer, 2002.

Part IV

One More Thing

Overview

Introduction

Now we'll look at a point we mentioned on our summary slide for Mahaney's Theorem but which we didn't have time to prove. Recall:

Mahaney's Theorem

If NP has sparse complete sets, then $P = NP$.

Does this hold if we replace "complete" with "hard"?

Yes:

Theorem 1.9

NP has sparse $\leq_m^p$-hard sets if and only if NP has sparse $\leq_m^p$-complete sets.

That is, the existence of sparse NP-hard sets and the existence of sparse NP-complete sets stand or fall together.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Proof

Since our theorem is "if and only if", we must show both directions:

Complete ⇒ Hard

If there are sparse $\leq_m^p$-complete sets in NP, then there are sparse $\leq_m^p$-hard sets in NP, obviously: Every set that is $\leq_m^p$-complete is also $\leq_m^p$-hard.

Hard ⇒ Complete

If NP has a $\leq_m^p$-hard sparse set, then it has a $\leq_m^p$-complete sparse set.

Proof

S is a sparse NP-hard set.
$f : SAT \rightarrow S$

We know $SAT \leq_m^p S$ since S is $\leq_m^p$-hard for NP, so let f be that P-computable reduction function.

$$S' = \{0^k \# y \mid k \geq 0 \wedge (\exists x \in SAT)[k \geq |x| \wedge f(x) = y]\}$$

S' is a padded encoding of $f(SAT)$.

If $0^k \# z \in S'$ then $z \in S$.
By definition of S', $z = f(x)$ where $x \in SAT$, so $f(x) \in S$.

Why is this necessary? Why not just $S' = \{x \# y \mid \ldots\}$?

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Proof

S is a sparse NP-hard set.
$f : SAT \rightarrow S$

We know $SAT \leq_m^p S$ since S is $\leq_m^p$-hard for NP, so let f be that P-computable reduction function.

$$S' = \{0^k \# y \mid k \geq 0 \wedge (\exists x \in SAT)[k \geq |x| \wedge f(x) = y]\}$$

S' is a padded encoding of $f(SAT)$.

If $0^k \# z \in S'$ then $z \in S$.
By definition of S', $z = f(x)$ where $x \in SAT$, so $f(x) \in S$.

Why is this necessary? Why not just $S' = \{x \# y \mid \ldots\}$?
That wouldn't be sparse. We rely on the fact that there is only one prefix 0^k for each k, whereas the binary representation of $x \in SAT$ is not sparse.

Algorithm for S'

$S' \in NP$. We show this by giving an algorithm:

Algorithm for S'

On input $0^k\#z$,

- Nondeterministically guess a string x of length at most k
- Nondeterministically guess a potential certificate of $x \in SAT$ (a complete assignment of the variables for the formula x)
- Accept if the guessed string and certificate is such that $f(x) = z$ and the certificate proves $x \in SAT$ (substituting the assignments gives a true formula).

Correctness: We accept an input only if it is in S': We've guessed the x that satisfies our requirements that $x \in SAT$, our given $k \geq |x|$, and $f(x)$ (the element of S we get from reducing from our element in SAT) is the input's z.

The algorithm is obviously nondeterministic and polynomial.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

End of Proof

$S' \in NP$

We've shown that given any sparse, NP-hard set S, there exists $S' \in NP$.

S' is NP-hard

Given $f : SAT \rightarrow S$, there exists a polynomial-time reduction $f' : SAT \rightarrow S'$ where $f'(x) = 0^{|x|} \# f(x)$.

S' is Sparse

And given S is sparse, S' is sparse. Why?

Thus we've shown that the existence of sparse NP-hard set S necessitates the existence of S', which is an NP-complete sparse set. Our proof is complete.

Amal Fahad, Chetan Bhole, Jonathan Gordon, Mehdi Manshadi

Summary

What have we covered?

- Sparse sets and tally sets
- Self-reducibility ($L = L(T^L)$, query words of length $\leq n-1$)
- Hardness and completeness
- The pruning technique
- If there is a tally set that is NP-hard, then $P = NP$
- If there is a sparse set that is $coNP$-hard, then $P = NP$
- Depth-first, breadth-first search methods
- Pseudocomplement
- If there is an NP-complete (or hard) sparse set, then $P = NP$
- The Hartmanis–Immerman–Sewelson Encoding
- E and NE
- Equivalent: $E = NE$, $NP - P$ contains no sparse sets, $NP - P$ contains no tally sets

Any questions? (One more proof to go if we have time!)

References

- Lane Hemaspaandra and Mitsunori Ogihara. *The Complexity Theorem Companion*. Springer: Springer, 2002.