Upper bound for *Circuit SAT*.

Sergey Nurk

Mathematics and Mechanics Faculty
Saint-Petersburg State University

October 2, 2009

Outline

Introduction

Algorithm

Upper Bound

Basic definitions

Definition

Formula (circuit) is called satisfiable iff there is a truth assignment for the variables that makes it evaluate to TRUE.

Basic definitions

Definition

Formula (circuit) is called satisfiable iff there is a truth assignment for the variables that makes it evaluate to TRUE.

Definition

To solve the *SAT*(*Circuit SAT*) problem for the CNF formula (circuit) means to determine whether it is satisfiable or not.

What upper bounds for general *SAT* do we know?

With respect to n

What upper bounds for general *SAT* do we know?

With respect to n

A lot of different results, but nothing better than $2^{\alpha n}$.

What upper bounds for general *SAT* do we know?

With respect to n

A lot of different results, but nothing better than $2^{\alpha n}$.

What do alternative approach give?

What upper bounds for general *SAT* do we know?

With respect to n

A lot of different results, but nothing better than $2^{\alpha n}$.

What do alternative approach give?

- $2^{0.30897m}$, where m is the number of clauses.

What upper bounds for general *SAT* do we know?

With respect to n

A lot of different results, but nothing better than $2^{\alpha n}$.

What do alternative approach give?

- $2^{0.30897m}$, where m is the number of clauses.
- $2^{0.10299l}$, where l is the total number of occurrences of all variables.

What about *Circuit SAT*?

What about *Circuit SAT*?

Practically nothing!

What about *Circuit SAT*?

Practically nothing!
Know no approaches for proving upper bounds in form of c^n ($c < 2$ is a constant) for the general case of the *Circuit SAT*.

What about *Circuit SAT*?

Practically nothing!
Know no approaches for proving upper bounds in form of c^n ($c < 2$ is a constant) for the general case of the *Circuit SAT*.
What about c^m, where m is the size of the circuit?

What about *Circuit SAT*?

Practically nothing!
Know no approaches for proving upper bounds in form of c^n ($c < 2$ is a constant) for the general case of the *Circuit SAT*.
What about c^m, where m is the size of the circuit?
Topic of this talk is an algorithm, that runs in time $O(2^{0.4058m})$.

Boolean Circuit

- Gates with fan-in 2.
- Single output.
- Consider full binary basis.

What about degenerate functions?

Proposition

Any gate that computes a function of one variable can be eliminated from the circuit.

Informal description

- Simplify the circuit.

Informal description

- Simplify the circuit.
- Modify if necessary.

Informal description

- Simplify the circuit.
- Modify if necessary.
- If circuit is not trivial then find 'good' variable.
- 'Split' on this variable.

Formal description

Algorithm CIRCUIT SAT

Input: circuit C.

Output: **True** if the circuit is satisfiable and **False** otherwise.

Formal description

Algorithm CIRCUIT SAT

Input: circuit C.

Output: **True** if the circuit is satisfiable and **False** otherwise.

1: **Return** $\textsc{Split}(\textsc{Reduce}(C))$.

Function SPLIT

Input: circuit C, obtained as a result of the function REDUCE.
Output: **True** if the circuit is satisfiable and **False** otherwise.

Function SPLIT

Input: circuit C, obtained as a result of the function REDUCE.
Output: **True** if the circuit is satisfiable and **False** otherwise.

1: If the output of the circuit C is a constant gate, then return its value.

Function SPLIT

Input: circuit C, obtained as a result of the function REDUCE.
Output: **True** if the circuit is satisfiable and **False** otherwise.

1: If the output of the circuit C is a constant gate, then return its value.
2: Choose a variable x that complies with one of the following conditions:

Function SPLIT

Input: circuit C, obtained as a result of the function REDUCE.
Output: **True** if the circuit is satisfiable and **False** otherwise.

1: If the output of the circuit C is a constant gate, then return its value.
2: Choose a variable x that complies with one of the following conditions:
   - outdegree no less than 3;

Function SPLIT

Input: circuit C, obtained as a result of the function REDUCE.
Output: **True** if the circuit is satisfiable and **False** otherwise.

1: If the output of the circuit C is a constant gate, then return its value.
2: Choose a variable x that complies with one of the following conditions:
   - outdegree no less than 3;
   - outdegree 2 and type-$\wedge$ direct successor;

Function SPLIT

Input: circuit C, obtained as a result of the function REDUCE.
Output: **True** if the circuit is satisfiable and **False** otherwise.

1: If the output of the circuit C is a constant gate, then return its value.
2: Choose a variable x that complies with one of the following conditions:

- outdegree no less than 3;
- outdegree 2 and type-$\wedge$ direct successor;

Remark. The function REDUCE ensures that such a variable exists.

Function SPLIT

Input: circuit C, obtained as a result of the function REDUCE.
Output: **True** if the circuit is satisfiable and **False** otherwise.

1: If the output of the circuit C is a constant gate, then return its value.
2: Choose a variable x that complies with one of the following conditions:
   - outdegree no less than 3;
   - outdegree 2 and type-$\wedge$ direct successor;

   Remark. The function REDUCE ensures that such a variable exists.
3: **Return** $\text{SPLIT}(\text{REDUCE}(C[x = 0]))$ **or** $\text{SPLIT}(\text{REDUCE}(C[x = 1]))$

Function Reduce

Input: a circuit C.
Output: a circuit C' which:

Function REDUCE

Input: a circuit C.
Output: a circuit C' which:

- is satisfiable iff the circuit C is satisfiable;
- is a constant gate or contains a variable that complies with the condition from the second step of the function $\text{SPLIT}(C)$;
- its size is no larger than the size of C.

Function REDUCE

Step 1. Eliminate constants and degenerate gates.

Function Reduce

Step 1. Eliminate constants and degenerate gates.
Step 2. Eliminate non-output gates with the outdegree 0.

Function Reduce

Step 1. Eliminate constants and degenerate gates.
Step 2. Eliminate non-output gates with the outdegree 0.
Step 3. If the output is a constant gate, then return this constant.

Function REDUCE

Step 1. Eliminate constants and degenerate gates.
Step 2. Eliminate non-output gates with the outdegree 0.
Step 3. If the output is a constant gate, then return this constant. If the output is a variable, return **True**.

Function Reduce

Step 1. Eliminate constants and degenerate gates.
Step 2. Eliminate non-output gates with the outdegree 0.
Step 3. If the output is a constant gate, then return this constant. If the output is a variable, return **True**.
Step 4. If there is a variable of outdegree more than 2, then return C.

Function Reduce

Step 1. Eliminate constants and degenerate gates.
Step 2. Eliminate non-output gates with the outdegree 0.
Step 3. If the output is a constant gate, then return this constant. If the output is a variable, return **True**.
Step 4. If there is a variable of outdegree more than 2, then return C.
Step 5. If there is a variable of outdegree 2 that has a type-$\wedge$ successor, then return C.

Function Reduce

Step 1. Eliminate constants and degenerate gates.
Step 2. Eliminate non-output gates with the outdegree 0.
Step 3. If the output is a constant gate, then return this constant. If the output is a variable, return **True**.
Step 4. If there is a variable of outdegree more than 2, then return C.
Step 5. If there is a variable of outdegree 2 that has a type-$\wedge$ successor, then return C.
Step 6. If there is a variable x of outdegree 2 then ???.

Function REDUCE

Step 1. Eliminate constants and degenerate gates.
Step 2. Eliminate non-output gates with the outdegree 0.
Step 3. If the output is a constant gate, then return this constant. If the output is a variable, return **True**.
Step 4. If there is a variable of outdegree more than 2, then return C.
Step 5. If there is a variable of outdegree 2 that has a type-$\wedge$ successor, then return C.
Step 6. If there is a variable x of outdegree 2 then ???.
Step 7. Replace an arbitrary top level gate of the circuit C (a gate whose parents are variables only) with a new input variable. Return REDUCE for the new circuit.

$\oplus$-chain

We say that for a gate G_0 from a circuit C there is a $\oplus$-*chain* of length k in G_0 iff there are k gates $G_1, \ldots, G_k$ in C, such that:

$\oplus$-chain

We say that for a gate G_0 from a circuit C there is a $\oplus$-*chain* of length k in G_0 iff there are k gates $G_1, \ldots, G_k$ in C, such that:

1. For $1 \leq i \leq k$, G_i is a type-$\oplus$ gate;

$\oplus$-chain

We say that for a gate G_0 from a circuit C there is a $\oplus$-*chain* of length k in G_0 iff there are k gates $G_1, \ldots, G_k$ in C, such that:

1. For $1 \leq i \leq k$, G_i is a type-$\oplus$ gate;
2. For $1 \leq i \leq k$, G_i is the only successor of the gate G_{i-1};

$\oplus$-chain

We say that for a gate G_0 from a circuit C there is a $\oplus$-*chain* of length k in G_0 iff there are k gates $G_1, \ldots, G_k$ in C, such that:

1. For $1 \leq i \leq k$, G_i is a type-$\oplus$ gate;
2. For $1 \leq i \leq k$, G_i is the only successor of the gate G_{i-1};
3. There is no $\oplus$-chain in G_k, i.e., G_k either is the circuit's output, or has outdegree no less than 2, or its only successor is a type-$\wedge$ gate.

Step 6.

If there is a variable x of outdegree 2 then consider $\oplus$-chains $P_1, \ldots, P_p$ and $R_1, \ldots, R_r$ that begin in its successors P_0 and R_0.

Step 6.

If there is a variable x of outdegree 2 then consider $\oplus$-chains $P_1, \ldots, P_p$ and $R_1, \ldots, R_r$ that begin in its successors P_0 and R_0.
Remark. p and r might equal zero in case if the corresponding $\oplus$-chains are empty.

Step 6.

If there is a variable x of outdegree 2 then consider $\oplus$-chains $P_1, \ldots, P_p$ and $R_1, \ldots, R_r$ that begin in its successors P_0 and R_0.
Remark. p and r might equal zero in case if the corresponding $\oplus$-chains are empty.
Two cases:

- $\oplus$-chains have no common elements.
- $\oplus$-chains have common elements.

Step 6.

Case 1. $\oplus$-chains have no common elements.

Step 6.

Case 1. $\oplus$-chains have no common elements.

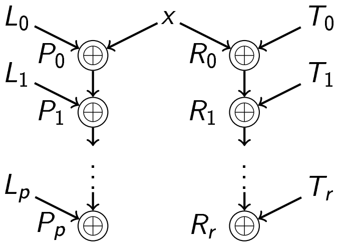

Figure: The case of non-intersecting $\oplus$-chains

Step 6.

Case 1. $\oplus$-chains have no common elements.

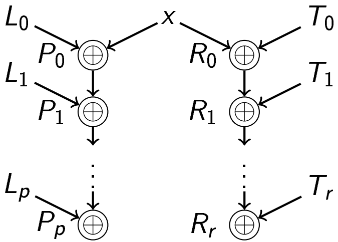

Figure: The case of non-intersecting $\oplus$-chains

Proposition

Assume there is no path from R_r to P_p. Then there is no path from x to L_i $(0 \leq i \leq p)$ and P_p is not the output of the circuit.

Step 6.

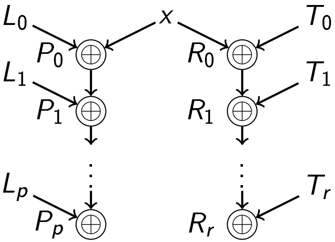

Figure: The case of non-intersecting $\oplus$-chains

Step 6.

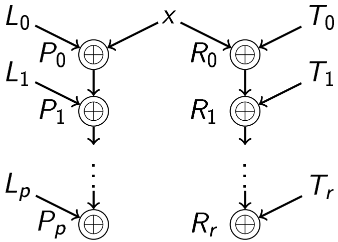

Figure: The case of non-intersecting $\oplus$-chains

Denote the value computed in P_p by y. Then

$$y = x \oplus L_0 \oplus L_1 \oplus \cdots \oplus L_p \oplus a \Longleftrightarrow x = y \oplus L_0 \oplus L_1 \oplus \cdots \oplus L_p \oplus a.$$

Step 6.

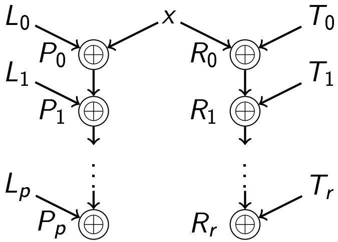

Figure: The case of non-intersecting $\oplus$-chains

Denote the value computed in P_p by y. Then

$$y = x \oplus L_0 \oplus L_1 \oplus \cdots \oplus L_p \oplus a \iff x = y \oplus L_0 \oplus L_1 \oplus \cdots \oplus L_p \oplus a.$$

The right side of second equation does not depend on x. It allows us to modify the circuit.

Step 6.

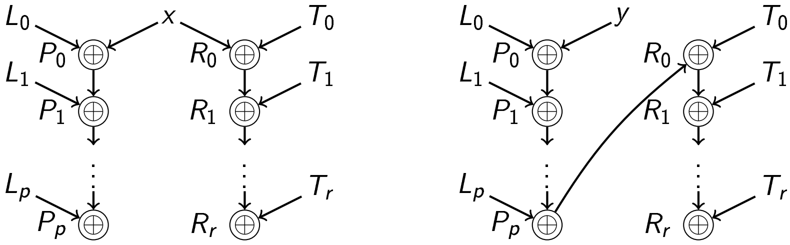

Figure: The case of non-intersecting $\oplus$-chains

Denote the value computed in P_p by y. Then

$$y = x \oplus L_0 \oplus L_1 \oplus \cdots \oplus L_p \oplus a \iff x = y \oplus L_0 \oplus L_1 \oplus \cdots \oplus L_p \oplus a.$$

The right side of second equation does not depend on x. It allows us to modify the circuit.

Step 6.

Case 2. P_k coincides with R_m for some $0 \leq k \leq p$ and $0 \leq m \leq r$.

Step 6.

Case 2. P_k coincides with R_m for some $0 \leq k \leq p$ and $0 \leq m \leq r$.

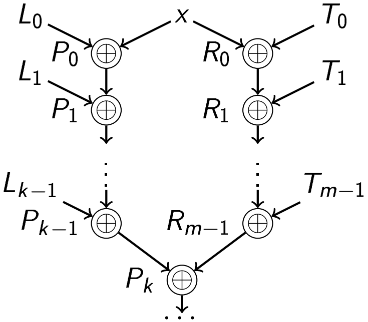

Figure: The case of intersecting $\oplus$-chains

Step 6.

Case 2. P_k coincides with R_m for some $0 \le k \le p$ and $0 \le m \le r$.

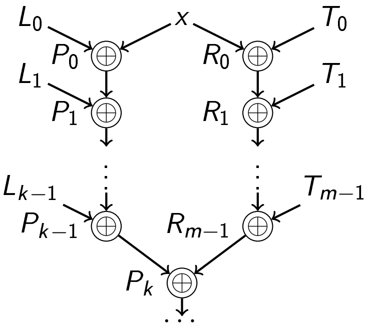

Figure: The case of intersecting $\oplus$-chains

In this case the value computed in P_k is

$$x \oplus x \oplus L_0 \oplus \cdots \oplus L_{k-1} \oplus T_0 \oplus \cdots \oplus T_{m-1} \oplus a$$

Step 6.

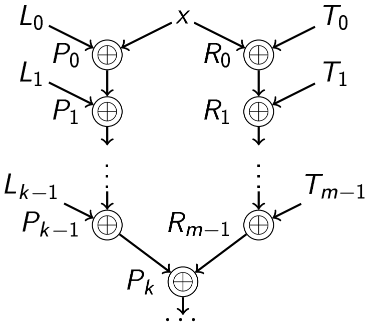

Figure: The case of intersecting $\oplus$-chains

The value computed in the gate P_k does not depend on x. It allows us to modify the circuit.

Step 6.

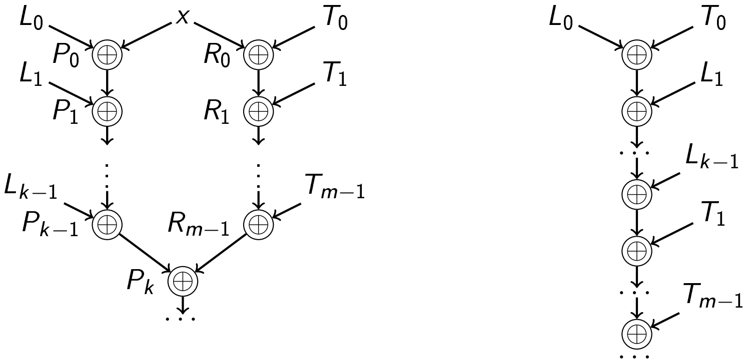

Figure: The case of intersecting $\oplus$-chains

The value computed in the gate P_k does not depend on x. It allows us to modify the circuit.

Step 6.

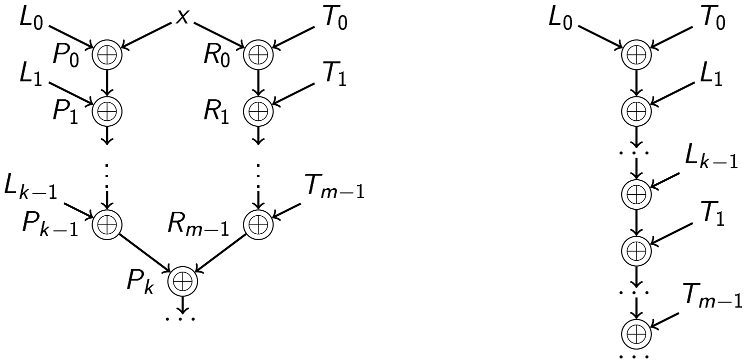

Figure: The case of intersecting $\oplus$-chains

The value computed in the gate P_k does not depend on x. It allows us to modify the circuit. Return REDUCE for the new circuit.

Splitting step

By x denote splitting variable.

Splitting step

By x denote splitting variable.

- If the outdegree of x is more than 2 then the substitution of both constants either reduces the size of the circuit at least by 3 or makes the circuit trivial.

Splitting step

By x denote splitting variable.

- If the outdegree of x is more than 2 then the substitution of both constants either reduces the size of the circuit at least by 3 or makes the circuit trivial.
- If the outdegree of x is 2 and one of its successors is a type-$\wedge$ gate then the substitution of some constant either reduces the size of the circuit at least by 3 or makes the circuit trivial and the substitution.

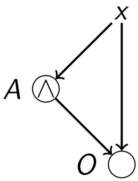

Figure: Interesting case.

Upper Bound

Theorem

The running time of the described algorithm is $O(2^{0.4058m})$, *where* m *is the size of the initial circuit.*

Upper Bound

Theorem

The running time of the described algorithm is $O(2^{0.4058m})$, where m is the size of the initial circuit.

Proof: Let us denote by $f(m)$ the maximum number of leaves in the tree of recursive calls of the described algorithm among all the circuits of size m.

Upper Bound

Theorem

The running time of the described algorithm is $O(2^{0.4058m})$*, where* m *is the size of the initial circuit.*

Proof: Let us denote by $f(m)$ the maximum number of leaves in the tree of recursive calls of the described algorithm among all the circuits of size m.

Previous slide brings us to the following recurrence relation:

$$f(m) \leq f(m-2) + f(m-3),$$

for $m \geq 3$.

Upper Bound

Theorem

The running time of the described algorithm is $O(2^{0.4058m})$, *where* m *is the size of the initial circuit.*

Proof: Let us denote by $f(m)$ the maximum number of leaves in the tree of recursive calls of the described algorithm among all the circuits of size m.

Previous slide brings us to the following recurrence relation:

$$f(m) \le f(m-2) + f(m-3),$$

for $m \ge 3$.

That leads to the upper bound.

$$f(m) \le poly(m)\tau^m \le O(2^{0.4058m}).$$

Thanks for your attention.