
Brainfuck on SAT Solver

[ledyba/bf2sat](https://github.com/ledyba/bf2sat)

ψ(プサイ) / PSI
@tikal ledyba.org

青春 × よさこい × ガールズストーリー

ハナヤマタ



ちりぬるを

サーッと サーッと

流れてく(ハイッ！)

うたかたの 花の香り

(ハイハイッ！)

ちりぬるを

SAT SAT

流れてく(ハイッ！)

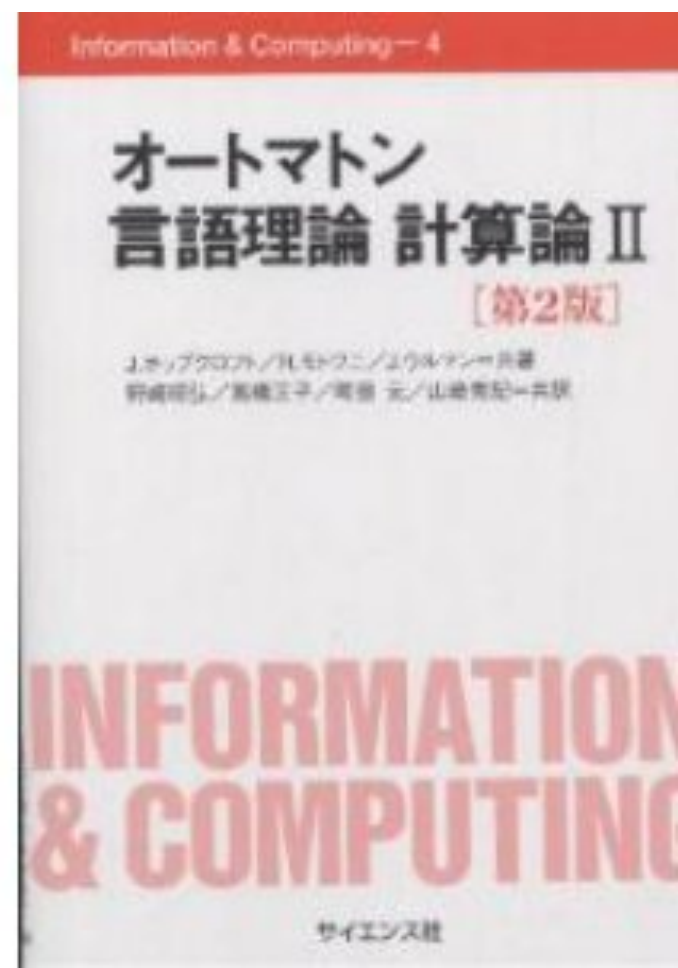
うたかたの 花の香り
(ハイハイッ！)

現代は一億総

SATソルバー

の時代

計算理論の おさらい



割と雑な
議論をするよ

SATソルバー

とは

何だったか

SAT isfiability problem

論理式

充足性問題

次の論理 A: 是(イエス)

x_1 と x_2 に $x_1 = \text{true}$

$x_2 = \text{false}$

$(x_1 \vee x_2) \wedge$

$(x_1 \vee \neg x_2) \wedge$

$(\neg x_1 \vee \neg x_2)$

ある論理式が
充足できるか
教えてくれるだけの
ソフトウェア

お、おう

それが
何なんだ

SATは

NP完全

SATは

NP完全

ある問題が
クラスNPに
属するとは

決定性Turing機械
を使って
多項式時間で
解の検算ができる

普通のコンピュータ
を使って
多項式時間で
解の検算ができる

答えは

$x_1 = \text{true}$

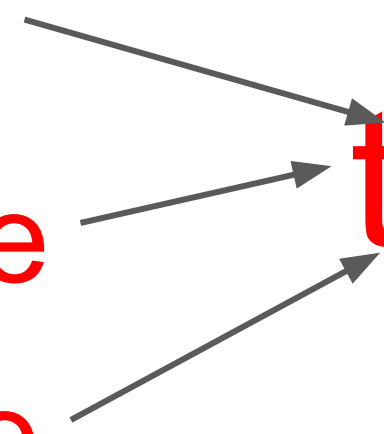
$x_2 = \text{false}$ だぞい！

$(x_1 \vee x_2) \wedge \text{true}$

$(x_1 \vee \neg x_2) \wedge \text{true}$

$(\neg x_1 \vee \neg x_2)$ true

true



```
graph LR; A["(x1 ∨ x2) ∧ true"] --> D["true"]; B["(x1 ∨ ¬x2) ∧ true"] --> D; C["(¬x1 ∨ ¬x2) true"] --> D;
```

その答えは
どうやって
出すんだ？

A.本質的に
しらみつぶし
 $O(2^n)$ の指数時間

A.本質的に
しらみつぶし
 $O(2^n)$ の指数時間
(たぶん。 $P \neq NP$ なら)

SATは

NP完全

SATは

NP完全

任意の

NP問題から

帰着可能

歸着可能？

I.Q Intelligent Qube の NP 完全性の証明

水 野 秀 一[†] 田 中 哲 朗^{††}

I.Q はよく知られたパズルゲームであるが、その計算量の議論はこれまで行われてこなかった。本論文では、I.Q のクリアにおいて重要な概念であるターン数に着目し、決められたターンにすべてのキューブを捕獲することができるかどうかというターン数判定問題が NP 完全であることを証明した。

7



1061400

7 Hits

1600 Points

11 Hits

8
115



$$\begin{aligned}
 & (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge \\
 & (x_1 \vee \neg x_2 \vee \neg x_3) \wedge \\
 & (\neg x_1 \vee \neg x_2 \vee x_3) \wedge \\
 & (x_1 \vee x_2 \vee \neg x_3) \wedge \\
 & (x_1 \vee x_2 \vee x_3)
 \end{aligned}$$

変換すると...

I.Q.が解ければ
SATも解ける
(帰着可能)

I.Q.の方が
SATより難しい

難しさ:

$SAT \leq I.Q.$

任意のNP問題

II $\wedge$

SATの問題

II $\wedge$

I.Q.の問題

I.Q.はNP

(コントローラを答え通りに動かすだけ)

任意のNP問題

$\parallel$

SATの問題

$\parallel$

I.Q.の問題

$\cup$

```
graph TD; A[任意のNP問題] --- B[||]; B --- C[SATの問題]; C --- D[||]; D --- E[I.Q.の問題]; F((U)); F -.- A; F -.- E;
```

難しさ:

$$\text{I.Q.} \leq \text{SAT} \leq \text{I.Q.}$$

難しさ:

I.Q. $\doteq$ SAT

難しさ:

I.Q. $\doteq$ SAT

I.Q.とSATはある意味で
同じぐらい難しい

SATやI.Q.と
同じぐらい
難しい問題が
NP完全

同様に...

SAT

≡ 3-PARTITION

≡ ぷよぷよ

≡ テトリス

任意のNP問題

$\Pi \wedge$ ← ここは？

SATの問題

$\Pi \wedge$

I.Q.の問題

$\cup$

自明

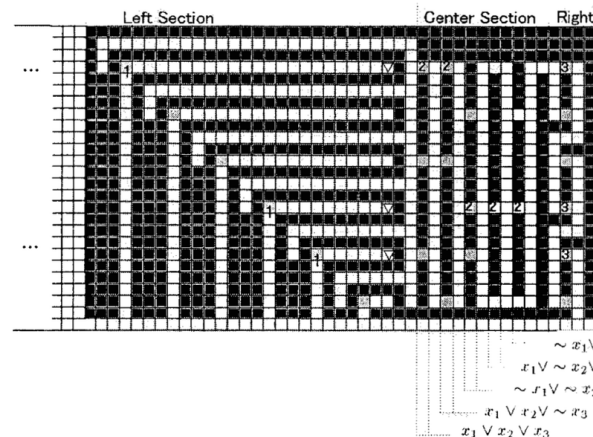


図 11 ターン数判定問題の解き方の例

任意のNP...

「NPなら何でも」

証明

難しそう



再掲

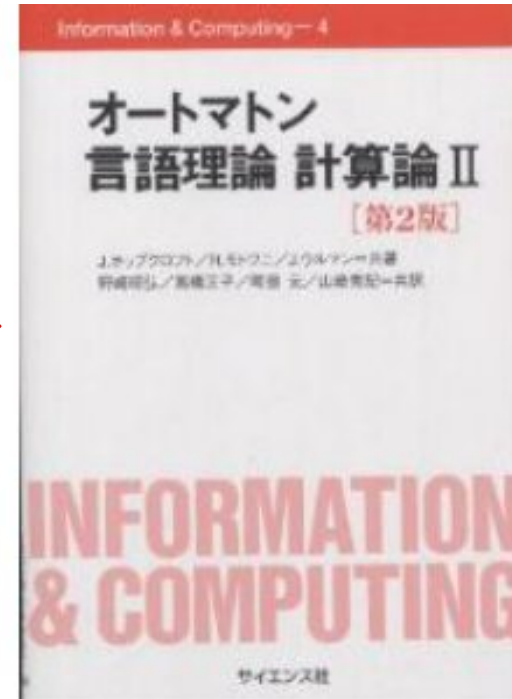
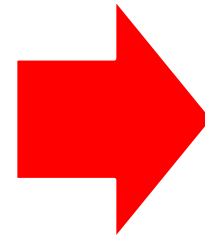
決定性Turing機械
を使って
多項式時間で
解の検算ができる

再掲

決定性Turing機械
を使って
多項式時間で
解の検算ができる

解の検証は
Turing機械≡
プログラムでやる
これをうまいこと使いたい！

多項式時間で解く
非決定性
Turing機械



SATに帰着(変換)

多項式時間で解く
非決定性

Brainfuckプログラム



SATに帰着(変換)

SATソルバーにシューツ！

github.com

/ledyba/bf2sat

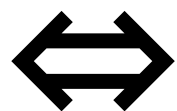
2*2を実行するだけの
プログラム

```
cabal run
Preprocessor: 0.1.0.0...
In-place: 1.0.0...
Preprocessor: brainfuck2sat-0.1.0.0...
** Brainfuck
-- Setting -
src:++[->++<]
ast:[ValInc,ValInc,LoopBegin 9,ValDec,PtInc,ValInc,ValInc,PtDec,LoopEnd 3]
in: [0, 0, 0, 0]
value-bits: 8
addr-bits:4
out-addr-bits:2
sim-steps:18
-- Create SAT problem --
To CNF...
313372 clauses, 4088273 literals
Aliasing...
84933 uniq predicates
writ to file
All done, have fun.
minisat sat.txt ans.txt
```

一つ一つの数字が論理変数
マイナス付きは¬がついたもの
同じ行の中ではV
違う行同士では^
だいたい30万行
全部で100万の論理変数

```
sat.txt
187784 50655 -50485 -50307 -50024 50996 -50845 -50690 51039 50060 0
187785 50655 -50485 -50307 -50024 50996 -50845 -50690 -51039 -45075 0
187786 50655 -50485 -50307 -50024 50996 -50845 -50690 -51039 -50060 0
187787 50655 -50485 -50307 -50024 50996 -50845 -50690 51040 45075 0
187788 50655 -50485 -50307 -50024 50996 -50845 -50690 51040 50060 0
187789 50655 -50485 -50307 -50024 50996 -50845 -50690 -51040 -45075 0
187790 50655 -50485 -50307 -50024 50996 -50845 -50690 -51040 -50060 0
187791 50655 -50485 -50307 -50024 50996 -50845 -50690 51041 45077 0
187792 50655 -50485 -50307 -50024 50996 -50845 -50690 51041 50062 0
187793 50655 -50485 -50307 -50024 50996 -50845 -50690 -51041 -45077 0
187794 50655 -50485 -50307 -50024 50996 -50845 -50690 -51041 -50062 0
187795 50655 -50485 -50307 -50024 50996 -50845 -50690 51042 45079 0
187796 50655 -50485 -50307 -50024 50996 -50845 -50690 51042 50064 0
187797 50655 -50485 -50307 -50024 50996 -50845 -50690 -51042 -45079 0
187798 50655 -50485 -50307 -50024 50996 -50845 -50690 -51042 -50064 0
187799 50655 -50485 -50307 -50024 50996 -50845 -50690 51043 45097 0
187800 50655 -50485 -50307 -50024 50996 -50845 -50690 51043 50082 0
187801 50655 -50485 -50307 -50024 50996 -50845 -50690 -51043 -45097 0
187802 50655 -50485 -50307 -50024 50996 -50845 -50690 -51043 -50082 0
187803 50655 -50485 -50307 -50024 50996 -50845 -50690 51044 45099 0
187804 50655 -50485 -50307 -50024 50996 -50845 -50690 51044 50084 0
```

Nステップ以内に
プログラムが
停止する



変換された
論理式が充足可能

```
psi@kaede3 ~/src/haskell/bf2sat
```

```
% minisat sat.txt ans.txt
```

```
WARNING: for repeatability, setting FPU to use double precision
```

```
===== [ Problem Statistics ] =====
```

```
|
| Number of variables:      84933
| Number of clauses:       307111
| Parse time:              0.30 s
| Eliminated clauses:      0.61 Mb
| Simplification time:     8.42 s
|
```

```
===== [ Search Statistics ] =====
```

```
| Conflicts | ORIGINAL | LEARNNT | Progress |
|           | Vars  Clauses Literals | Limit  Clauses Lit/Cl |
```

```
=====
=====
```

```
restarts      : 1
conflicts     : 0 (0 /sec)
decisions     : 1 (0.00 % random) (0 /sec)
propag        (619 /sec)
confl         (-nan % deleted)
```

```
Memory
CPU ti
```

充足可能
その時の論理式の
真偽の割り当ても
求めてくれる

```
SATISFIABLE
```

```
psi@kaede3 ~/src/haskell/bf2sat
```

-- Result --

Source:++[->++<]

Input: [0, 0, 0, 0]

Estimated Input: [0, 0, 0, 0]

Estimated IDs:

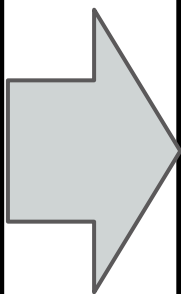
0: ID {getPC = 0, getMem = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
1: ID {getPC = 1, getMem = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
2: ID {getPC = 2, getMem = [2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
3: ID {getPC = 3, getMem = [2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
4: ID {getPC = 4, getMem = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
5: ID {getPC = 5, getMem = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 1, getIC = 0, getOC = 0, getOut = []}
6: ID {getPC = 6, getMem = [1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 1, getIC = 0, getOC = 0, getOut = []}
7: ID {getPC = 7, getMem = [1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 1, getIC = 0, getOC = 0, getOut = []}
8: ID {getPC = 8, getMem = [1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
9: ID {getPC = 3, getMem = [1,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
10: ID {getPC = 4, getMem = [0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
11: ID {getPC = 5, getMem = [0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 1, getIC = 0, getOC = 0, getOut = []}
12: ID {getPC = 6, getMem = [0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 1, getIC = 0, getOC = 0, getOut = []}
13: ID {getPC = 7, getMem = [0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 1, getIC = 0, getOC = 0, getOut = []}
14: ID {getPC = 8, getMem = [0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
15: ID {getPC = 9, getMem = [0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
16: ID {getPC = 9, getMem = [0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
17: ID {getPC = 9, getMem = [0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0], getPT = 0, getIC = 0, getOC = 0, getOut = []}
All done, have fun.

実行時のメモリ列を
SATソルバーが生成する

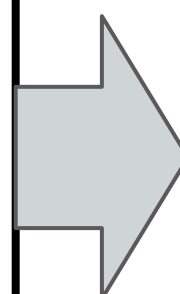
答えである4が求まってる

NPの問題を
解かせるには
非決定性の
機能が必要

入
力

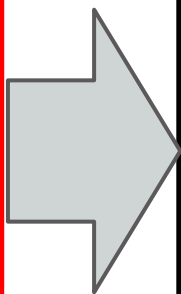


Brainfuck
プログラム

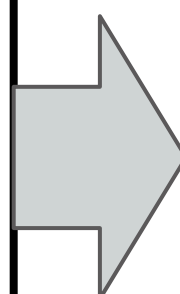


出
力

入力



Brainfuck
プログラム



出力

なぜか

答えが降ってくる

ふしぎな魔法の

入力テープ《オラクル》

...があると想像する

入力から
読んだ答えが
正しくないと
無限ループする
プログラムを書く

入力が2でないと
絶対に止まらない
プログラム

入力を「？」に
しておく

-- Result --

Source: , [> ++ < -] > ---- []

Input: [(?), 0, 0, 0]

Estimated Input: [2, 0, 0, 0]

Estimated IDs:

0: ID {getPC = 0, getMem = [0, 0, 0, 0], getOut = [0, 0, 0, 0]}	getOut = []
1: ID {getPC = 1, getMem = [2, 0, 0, 0], getOut = [0, 0, 0, 0]}	getOut = []
2: ID {getPC = 2, getMem = [2, 0, 0, 0], getOut = [0, 0, 0, 0]}	getOut = []
3: ID {getPC = 3, getMem = [2, 0, 0, 0], getOut = [0, 0, 0, 0]}	getOut = []
4: ID {getPC = 4, getMem = [2, 1, 0, 0], getOut = [0, 0, 0, 0]}	getOut = []

SATソルバが
答えを推測する

これで
NP完全な問題を
解こうか

プログラムが
長い

⇔ 論理式がデカい

メモリの的に
厳しい
(128GBのXEON
でも足りない)

素因数分解で
今日の所は
勘弁しといて
下さい(˘ ˘)

素数かどうかの判定は
NPより簡単なクラスPに属することが
2004年にやっと分かった

PRIMES is in P

Manindra Agrawal

Neeraj Kayal

Nitin Saxena*

Department of Computer Science & Engineering

Indian Institute of Technology Kanpur

Kanpur-208016, INDIA

Email: {manindra,kayaln,nitinsa}@iitk.ac.in

2つの入力の積が
15でないと
停止しないコード

```
-- Result --
```

```
Source: , > , <
```

```
[> [> + > + < < - ] > > [ < < + > > -
```

```
> > ----- [ ]
```

```
Input: [ ( ? ) , ( ? ) ]
```

```
Estimated Input: [ 3 , 5 ]
```

入力を
空白にするとく

答えは3と5

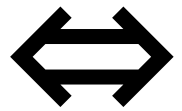
どうやって

SATに

変換する??

実は論理式は
割と
プログラム

コードが
実行される



部分論理式が
真になる

プログラム:

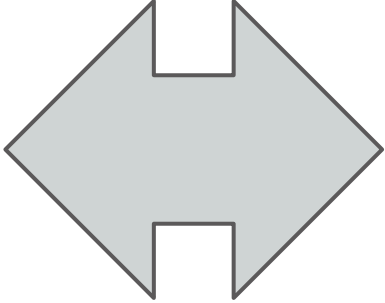
逐次実行

分岐

ループ

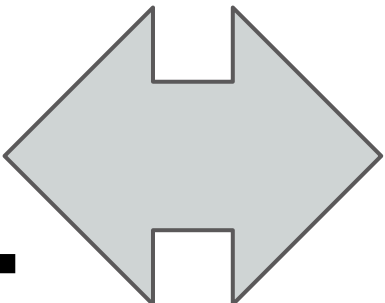
逐次実行 $\Leftrightarrow \wedge$

execA; A $\wedge$ B $\wedge$
execB; C
execC;



AもBもCも実行される
 $\Leftrightarrow$ AもBもCも真になる

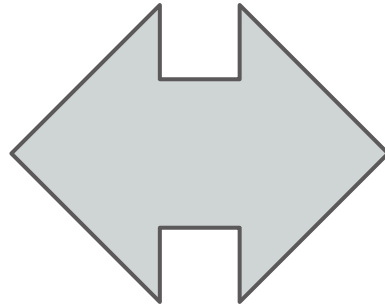
分岐 $\Leftrightarrow \vee$

if(c) a;
else b;  $(C \wedge A) \vee$
 $(\neg C \wedge B)$

CならA、 $\neg C$ ならB

ループ $\Leftrightarrow$ コピペ

```
for(*){  
  a;  
}
```



```
A ^ A ^  
A ^ ....
```

固定回数ループしか
書けない！！



トデス子\ @todesking · Jul 10



SATソルバーってチューリング完全なのか

[View translation](#)

FAVORITES

2



2:37 AM - 10 Jul 2015 · [Details](#)

固定回しか
ループ
できない

無限ループは
出来ない

SATソルバは
チューリング
完全ではない

NPの定義は
何だったか

再掲

決定性Turi
を使って

ある定数N回で
プログラムが
止まることが
保証できる

多項式時間で
解の検算ができる

十と一の計算:

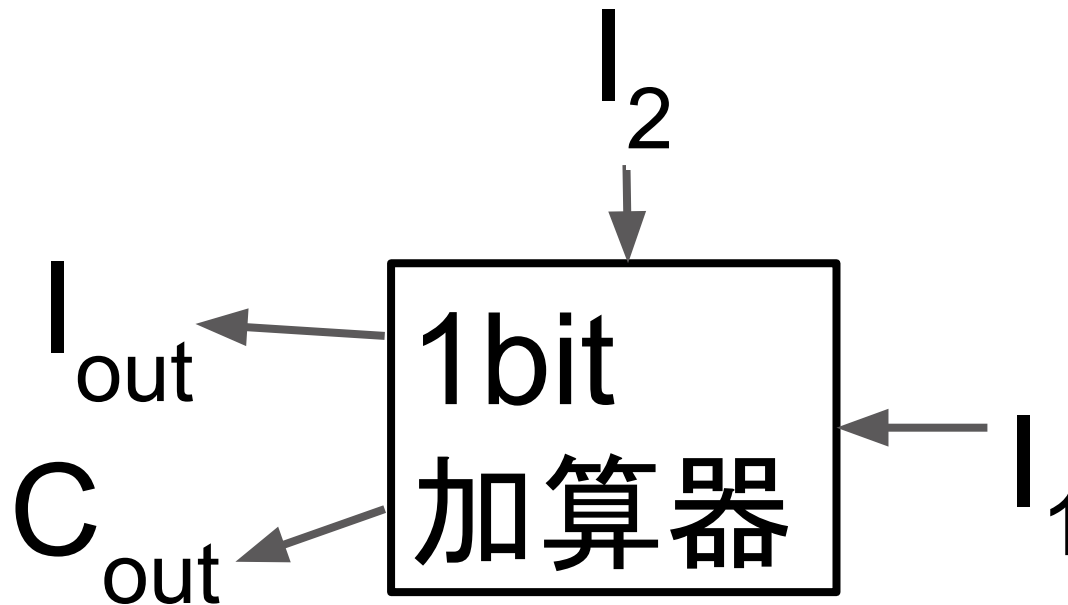
論理式は

論理回路

だと思っとく

1ビット半加算機

$$(I_1 \wedge (I_2 \neq I_{\text{out}}) \wedge (I_2 = C_{\text{out}})) \vee$$
$$(\neg I_1 \wedge (I_2 = I_{\text{out}}) \wedge (\neg C_{\text{out}}))$$



レジスタに対応する
論理変数(時刻t)

± 1

1bit
加算器

1bit
加算器

1bit
加算器

レジスタに対応する
論理変数(時刻t+1)

キャリーの遅延とか
気にしなくてOK
サブの論理変数を使
うと書きやすい

任意のNP問題

証明の円環が
閉じた

I.Q.の問題

任意のNPな
問題は
SATに変換可能

なので

超速いSATが
作れれば
他の問題も
超速く解ける

$O(2^n)$ だけど、
現実的は超速

$n=1000000$

ぐらい余裕

The international SAT Competitions web page

SAT 2014 competition

[Anton Belov](#), [Daniel Diepold](#), [Marijn Heule](#), [Matti Järvisalo](#)

[Pete Manolios](#), [Lakhdar Sais](#) and [Peter Stuckey](#)

[Descriptions of the solvers and benchmarks](#)

[Application](#), [Hard combinatorial](#), [Random](#)

Source code available in [EDACC](#)

Application			Hard combinatorial	
Gold	Silver	Bronze	Gold	Silver
Core solvers				
Lingeling	SWDiA5BY	Riss BlackBox	glueSplit_clasp	Lingeling
minisat_blbd	Riss BlackBox	SWDiA5BY	SparrowToRiss	CCAnr+glucose
Lingeling (druplig)	glucose	SWDiA5BY	Riss BlackBox	Lingeling (druplig)
Core solvers, Parallel				
Plingeling	PeneLoPe	Treengeling	Treengeling	Plingeling
Minisat hack				
MiniSat_HACK_999ED	minisat_blbd	ROKKminisat		

<http://www.satcompetition.org/>

この業界(?)の
アセンブラ的
存在

私と一緒に、

論理式充足

しませんか？

