Efficient Sampling of SAT Solutions for Testing

Rafael Dutra, Kevin Laeufer, Jonathan Bachrach, Koushik Sen
EECS Department
UC Berkeley

International Conference in Software Engineering
May 31, 2018

Problem definition

Input: Boolean constraint (SAT formula)

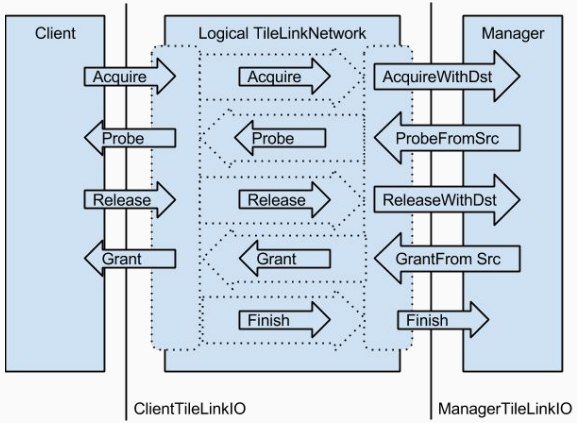
$$
\begin{aligned}
&(x1 \lor x4)\\
\land\ &(x1 \lor \neg x3 \lor \neg x8)\\
\land\ &(x1 \lor x8 \lor x6)\\
\land\ &(x2 \lor x5)\\
\land\ &(\neg x7 \lor \neg x3 \lor x9)\\
\land\ &(\neg x7 \lor x8 \lor \neg x9)\\
\land\ &(x7 \lor x8 \lor \neg x10)\\
\land\ &(x7 \lor x10 \lor \neg x6)
\end{aligned}
$$

Goal: Quickly generate lots of solutions that satisfy the constraint

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
σ_0	1	0	0	0	1	0	0	0	1	0
σ_1	0	0	0	1	1	0	0	1	1	0
σ_2	1	1	0	0	1	0	0	0	1	0
σ_3	0	1	0	1	1	0	0	1	1	0
σ_4	1	0	1	0	1	0	0	0	1	0
σ_5	1	1	1	0	1	0	0	0	1	0

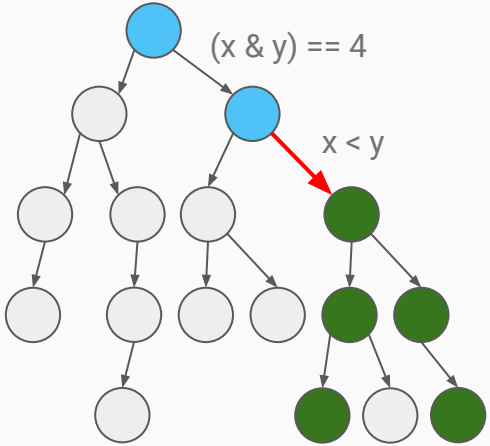
Motivation

- Thoroughly exercising some target functionality
- Constrained-Random Verification




```
void main(int4 x, int4 y) {
        if ((x & y) == 4) {
                if (x < y) {
                        interesting(x, y);
                }
        }
}
```

- Symbolic execution



Example

```
void main(int4 x, int4 y) {
    if ((x & y) == 4) {
        if (x < y) {
            interesting(x, y);
        }
    }
}
```

Example

```
void main(int4 x, int4 y) {
    if ((x & y) == 4) {
        if (x < y) {
            interesting(x, y);
        }
    }
}
```

Path Constraint

$$((x \,\&\, y) == 4) \land (x < y)$$

Example

```
void main(int4 x, int4 y) {
    if ((x & y) == 4) {
        if (x < y) {
            interesting(x, y);
        }
    }
}
```

int4 **x**;

x3	x2	x1	x0

int4 **y**;

y3	y2	y1	y0

Path Constraint

$$((x \,\&\, y) == 4) \wedge (x < y)$$

Bit-blast

Example

```
void main(int4 x, int4 y) {
    if ((x & y) == 4) {
        if (x < y) {
            interesting(x, y);
        }
    }
}
```

int4 **x**;

x3	x2	x1	x0

int4 **y**;

y3	y2	y1	y0

Path Constraint

$$((x \,\&\, y) == 4) \land (x < y)$$

Bit-blast

Boolean formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$

$$
\begin{aligned}
&(\neg x0 \lor \neg y0)\\
\land\ &(\neg x1 \lor \neg y1)\\
\land\ &x2\\
\land\ &y2\\
\land\ &(\neg x3 \lor \neg y3)\\
\land\ &(\quad (x3 \land \neg y3)\\
&\quad \lor (x3{=}y3 \land \neg x2 \land y2)\\
&\quad \lor (x3{=}y3 \land x2{=}y2 \land \neg x1 \land y1)\\
&\quad \lor (x3{=}y3 \land x2{=}y2 \land x1{=}y1 \land \neg x0 \land y0)\\
&)
\end{aligned}
$$

QuickSampler

QuickSampler

Our goals:

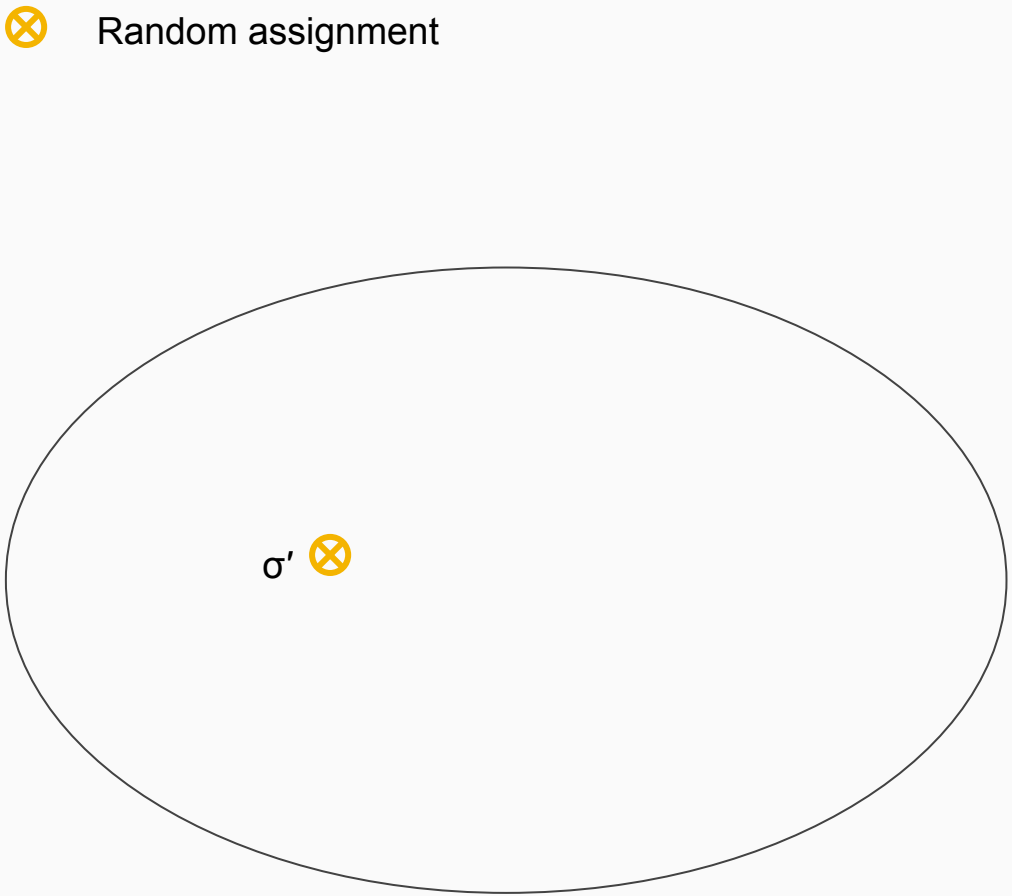
- Samples should satisfy the formula >50% of the times
- Generate samples >100x faster than other techniques
- Sampling should be close to uniform

Our approach:

- Compute patterns of bit flips which preserve satisfiability
- Combine those bit flip patterns to generate lots of samples

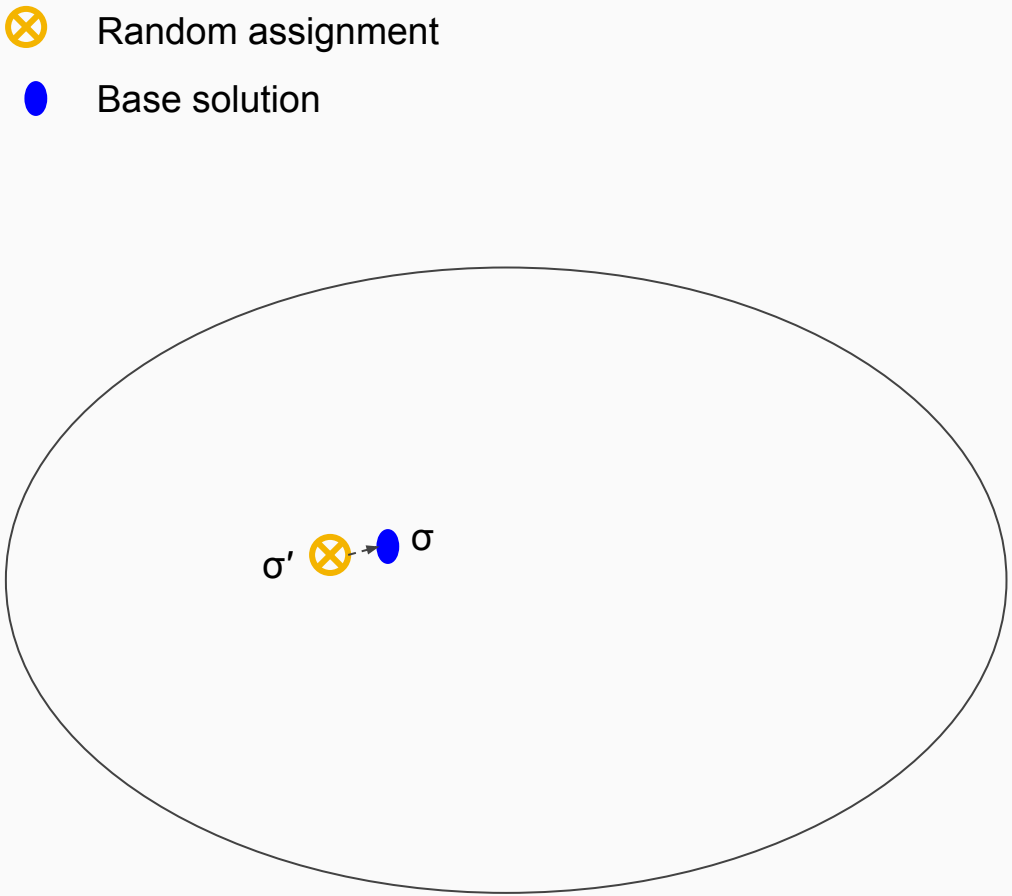
QuickSampler Algorithm

- Start by generating a random assignment σ’



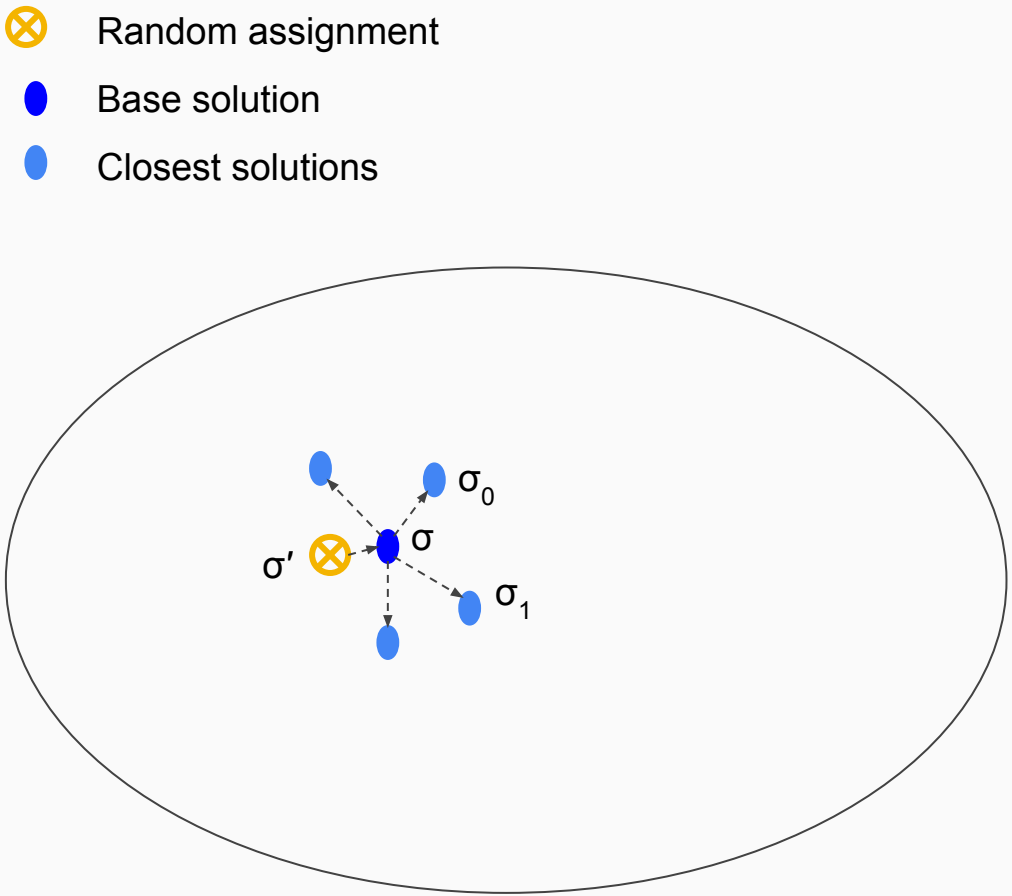
QuickSampler Algorithm

- Start by generating a random assignment σ'
- Use MAX-SAT to find the closest solution σ to σ'



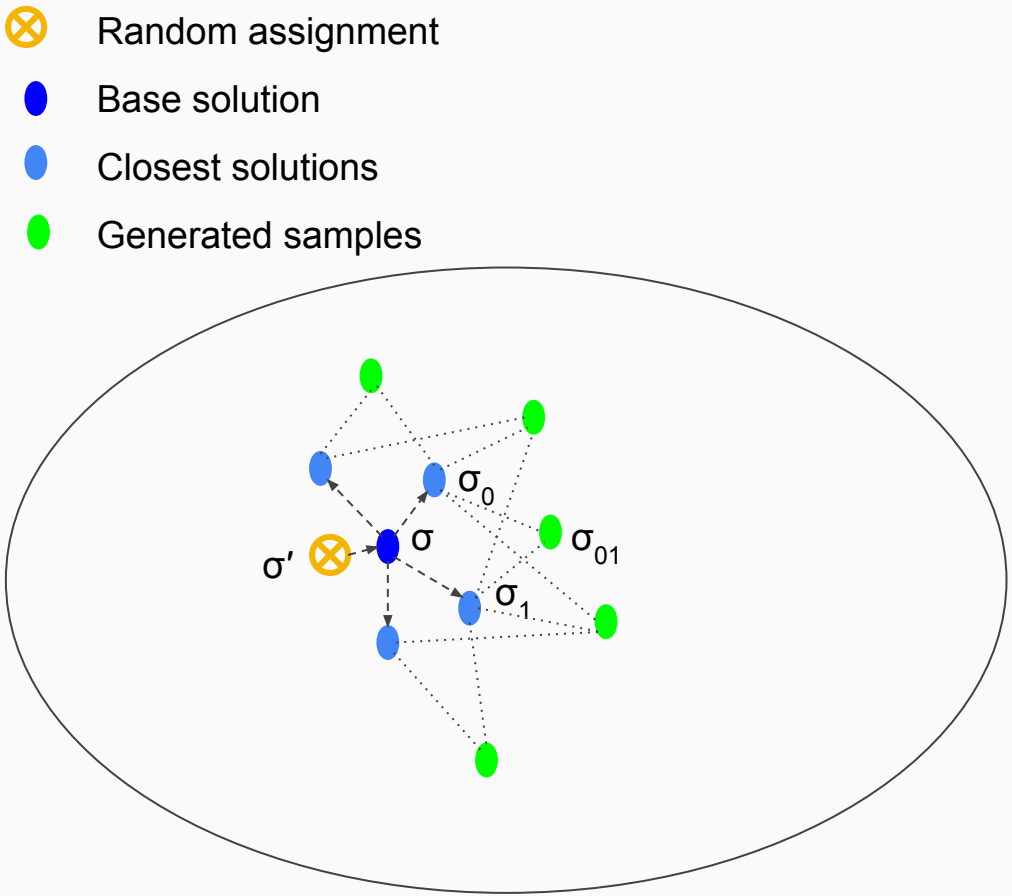
QuickSampler Algorithm

- Start by generating a random assignment σ'
- Use MAX-SAT to find the closest solution σ to σ'
- From σ, use MAX-SAT to find the closest solutions that flip some bits (such as σ_0 and σ_1)



QuickSampler Algorithm

- Start by generating a random assignment σ'
- Use MAX-SAT to find the closest solution σ to σ'
- From σ, use MAX-SAT to find the closest solutions that flip some bits (such as σ_0 and σ_1)
- Combine those mutation to generate new samples (such as σ_{01})



Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$

Formula **φ(x0,x1,x2,x3,y0,y1,y2,y3)**

	x0	x1	x2	x3	y0	y1	y2	y3
Random assignment σ’	0	0	1	0	1	1	0	0

Formula **φ(x0,x1,x2,x3,y0,y1,y2,y3)**

	x0	x1	x2	x3	y0	y1	y2	y3
Random assignment σ’	0	0	1	0	1	1	0	0
Solution σ	0	0	1	0	1	1	1	0

MAX-SAT

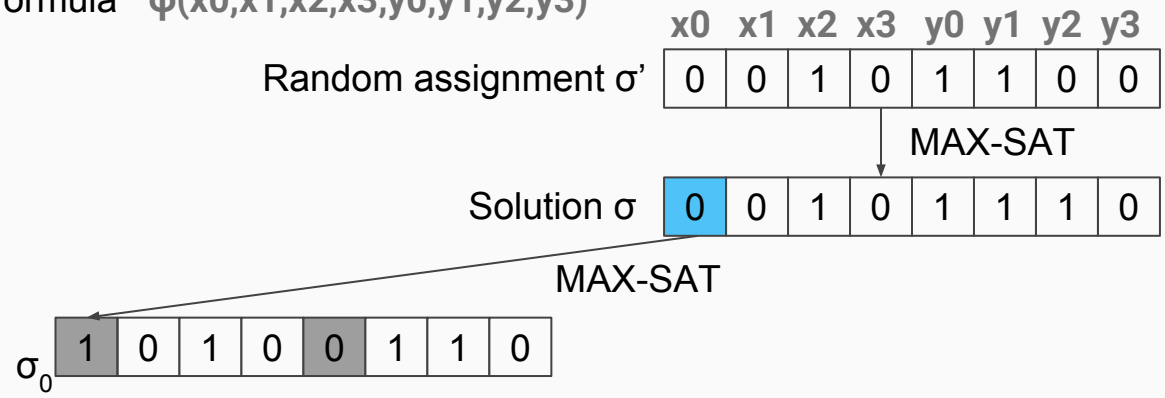
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$

	x0	x1	x2	x3	y0	y1	y2	y3
Random assignment σ’	0	0	1	0	1	1	0	0
Solution σ	0	0	1	0	1	1	1	0

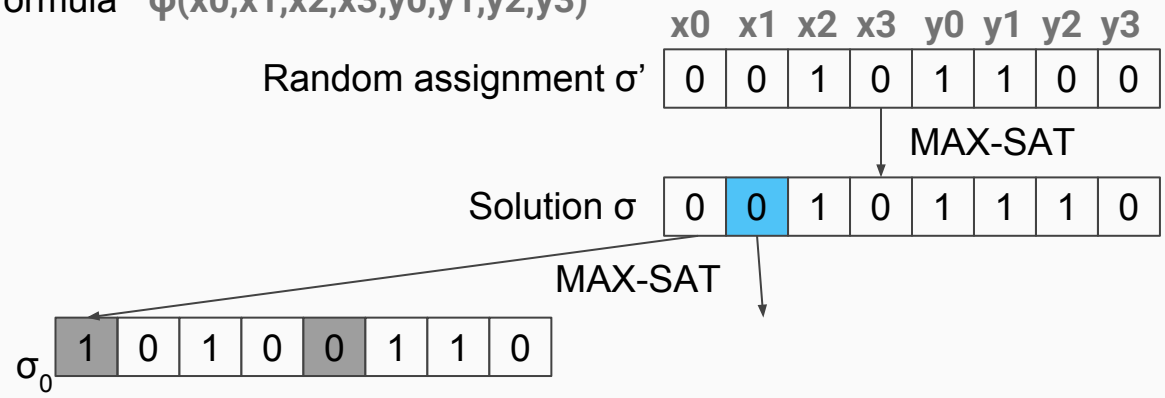
MAX-SAT

MAX-SAT

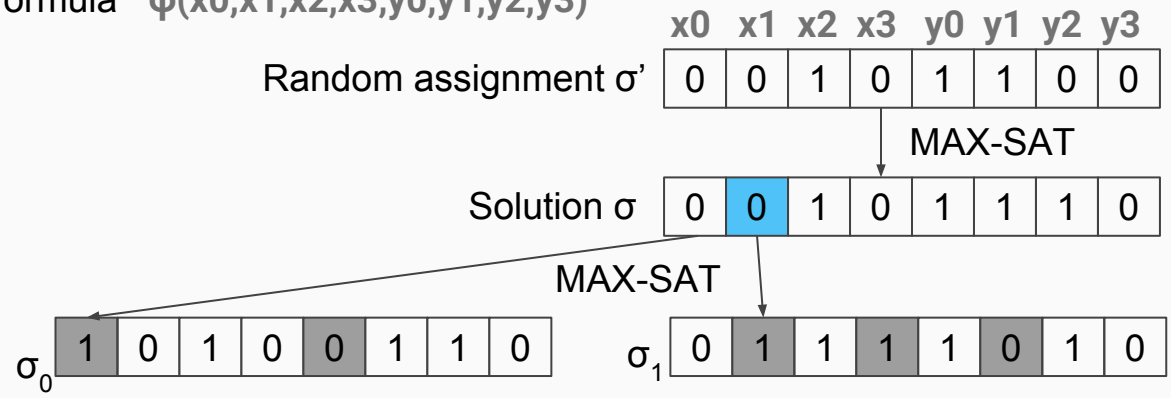
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



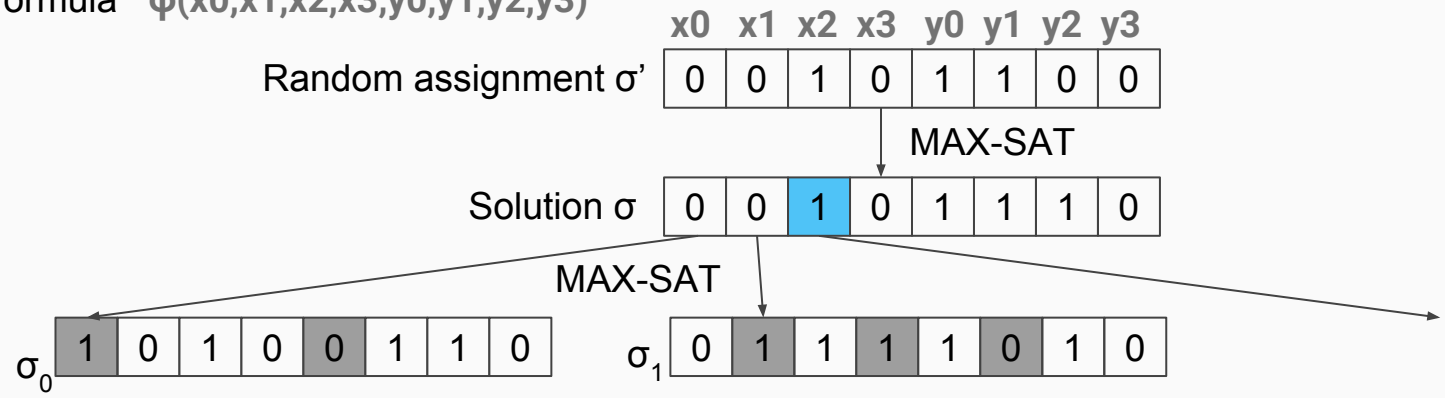
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



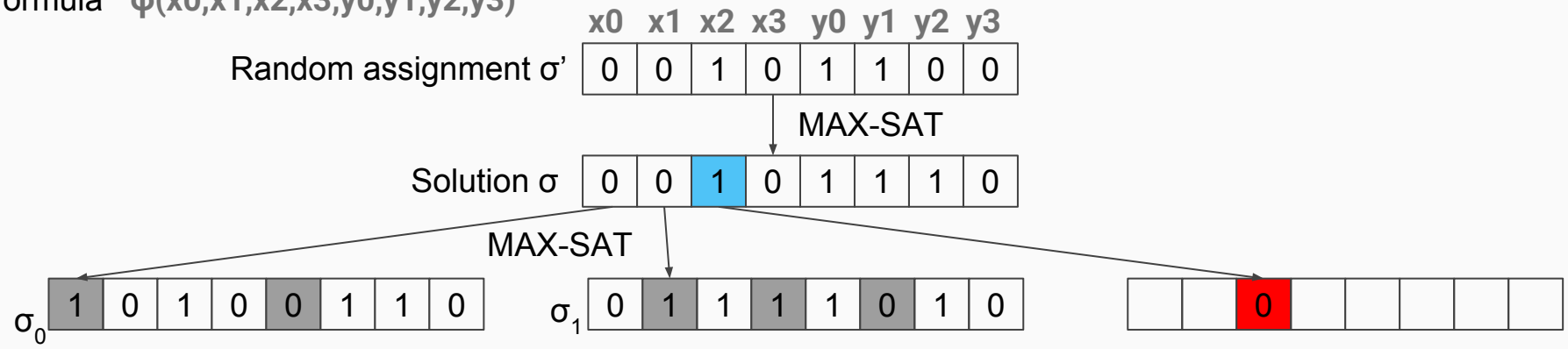
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



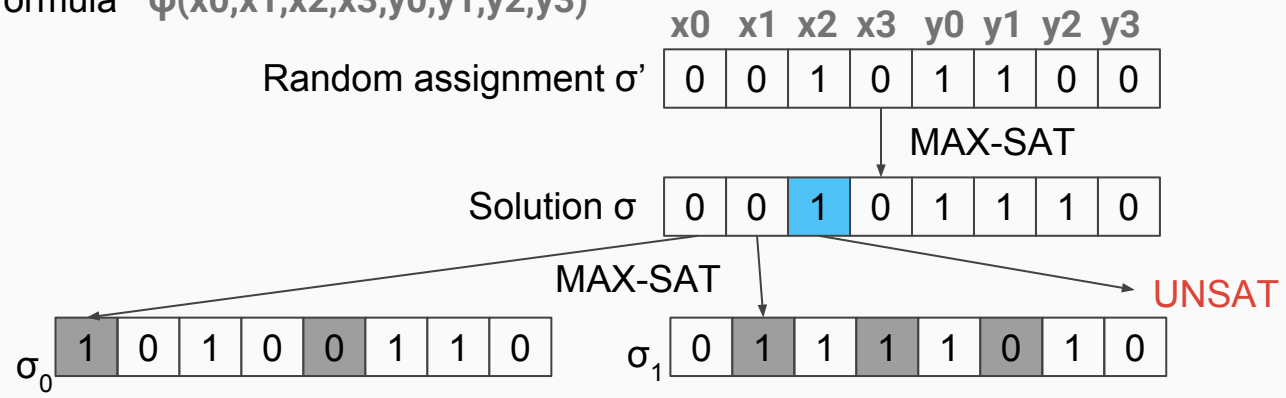
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



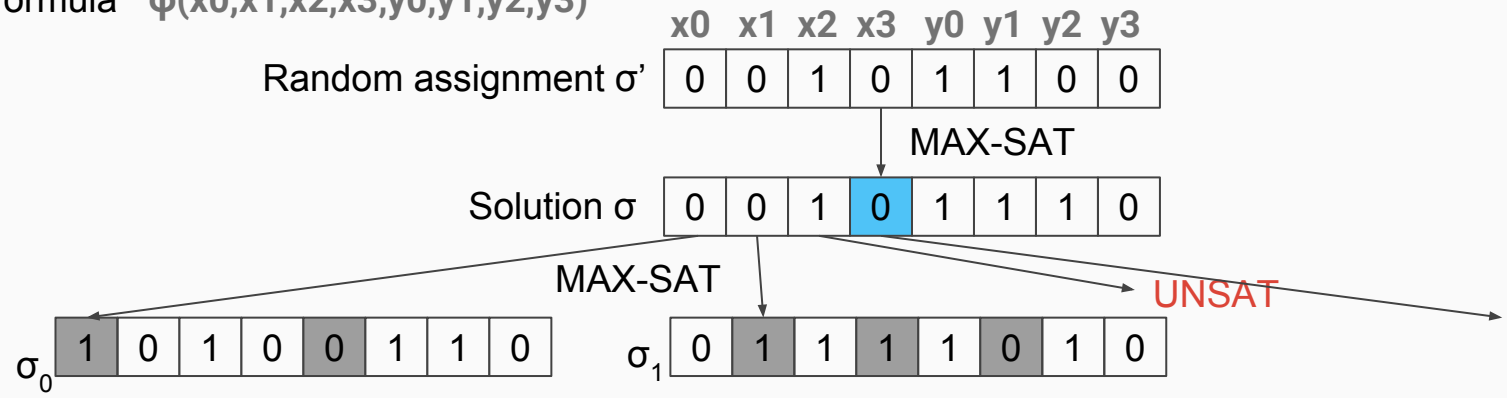
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



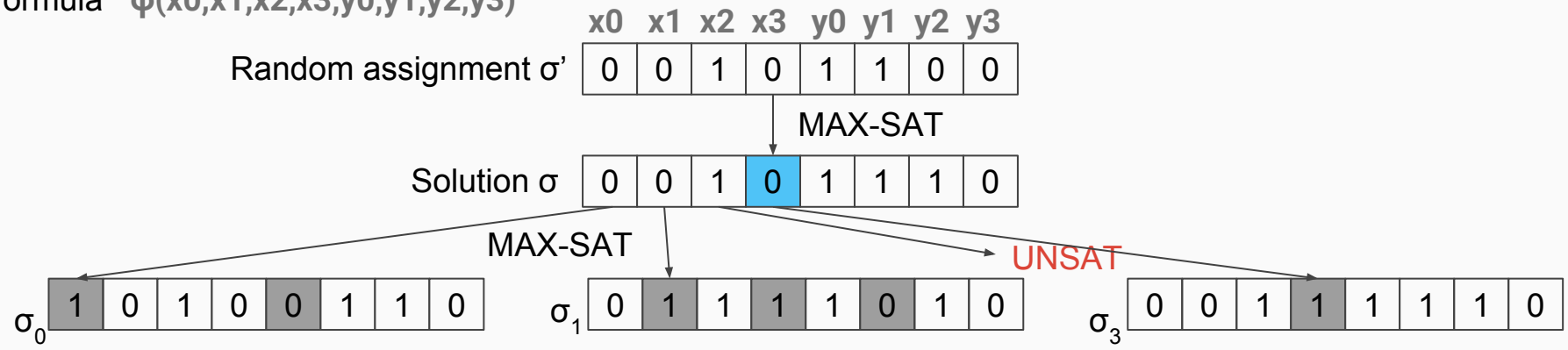
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



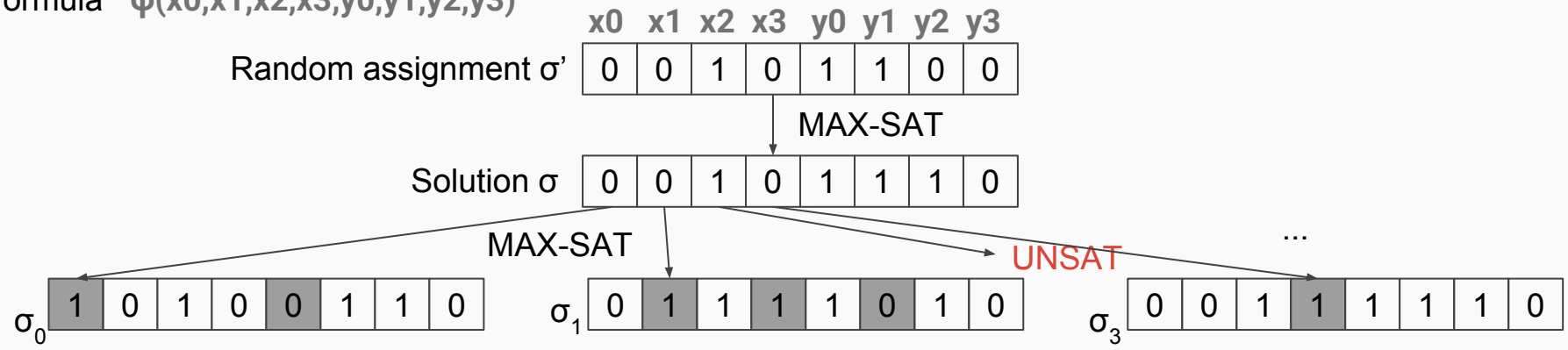
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$

	x0	x1	x2	x3	y0	y1	y2	y3
Random assignment σ’	0	0	1	0	1	1	0	0

MAX-SAT

Solution σ	0	0	1	0	1	1	1	0

MAX-SAT

UNSAT

...

σ_0	1	0	1	0	0	1	1	0
δ_0	1	0	0	0	1	0	0	0

$= \sigma \oplus \sigma_0$

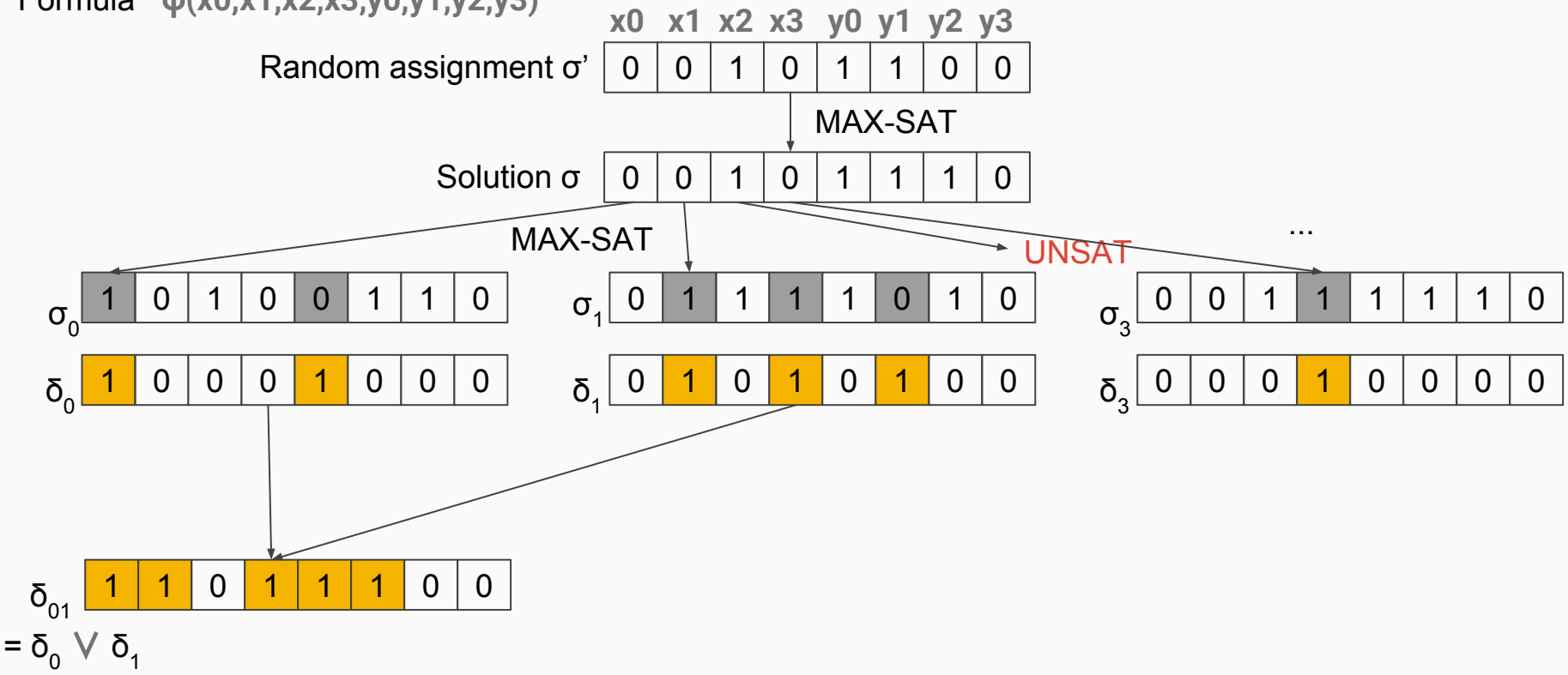
σ_1	0	1	1	1	1	0	1	0
δ_1	0	1	0	1	0	1	0	0

$= \sigma \oplus \sigma_1$

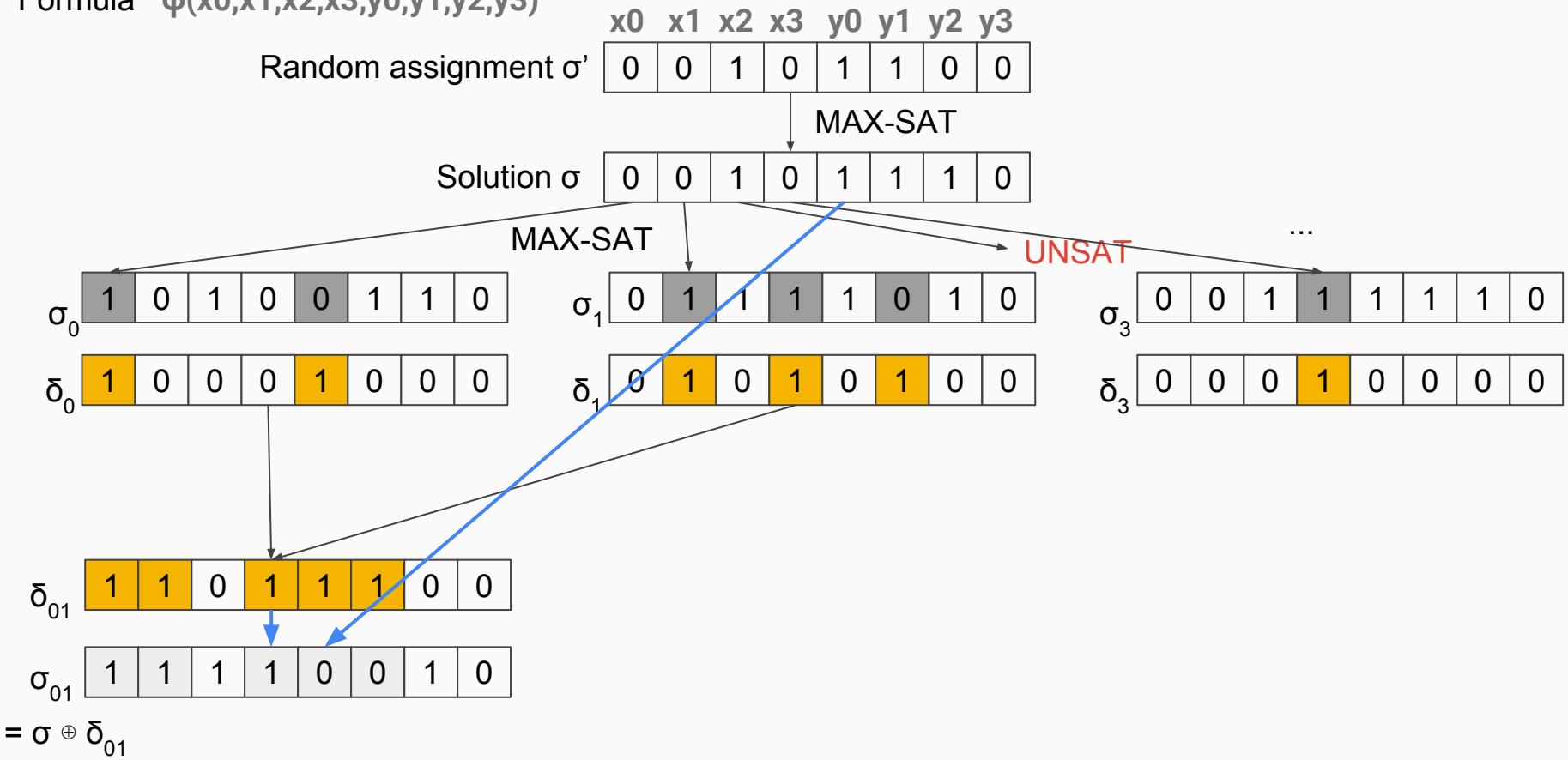
σ_3	0	0	1	1	1	1	1	0
δ_3	0	0	0	1	0	0	0	0

$= \sigma \oplus \sigma_3$

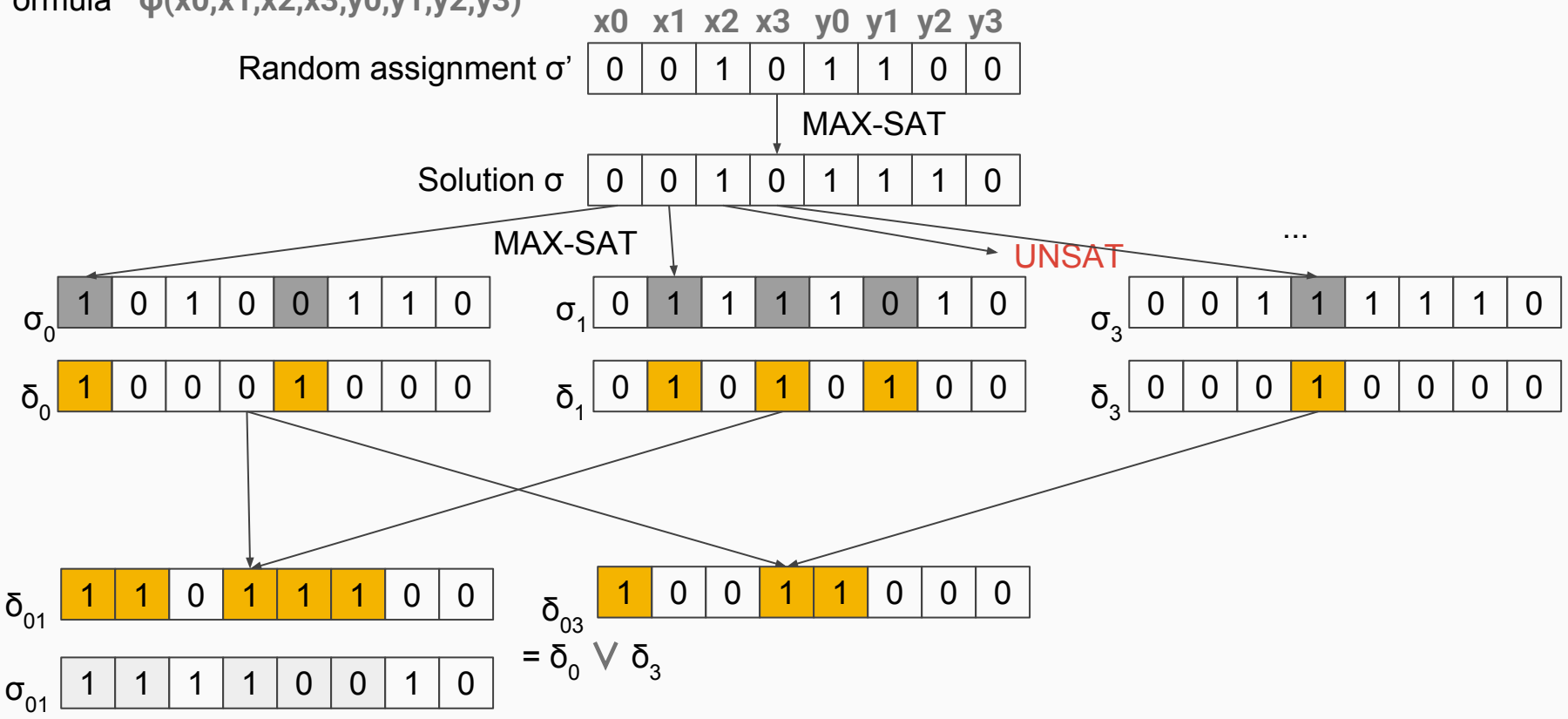
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



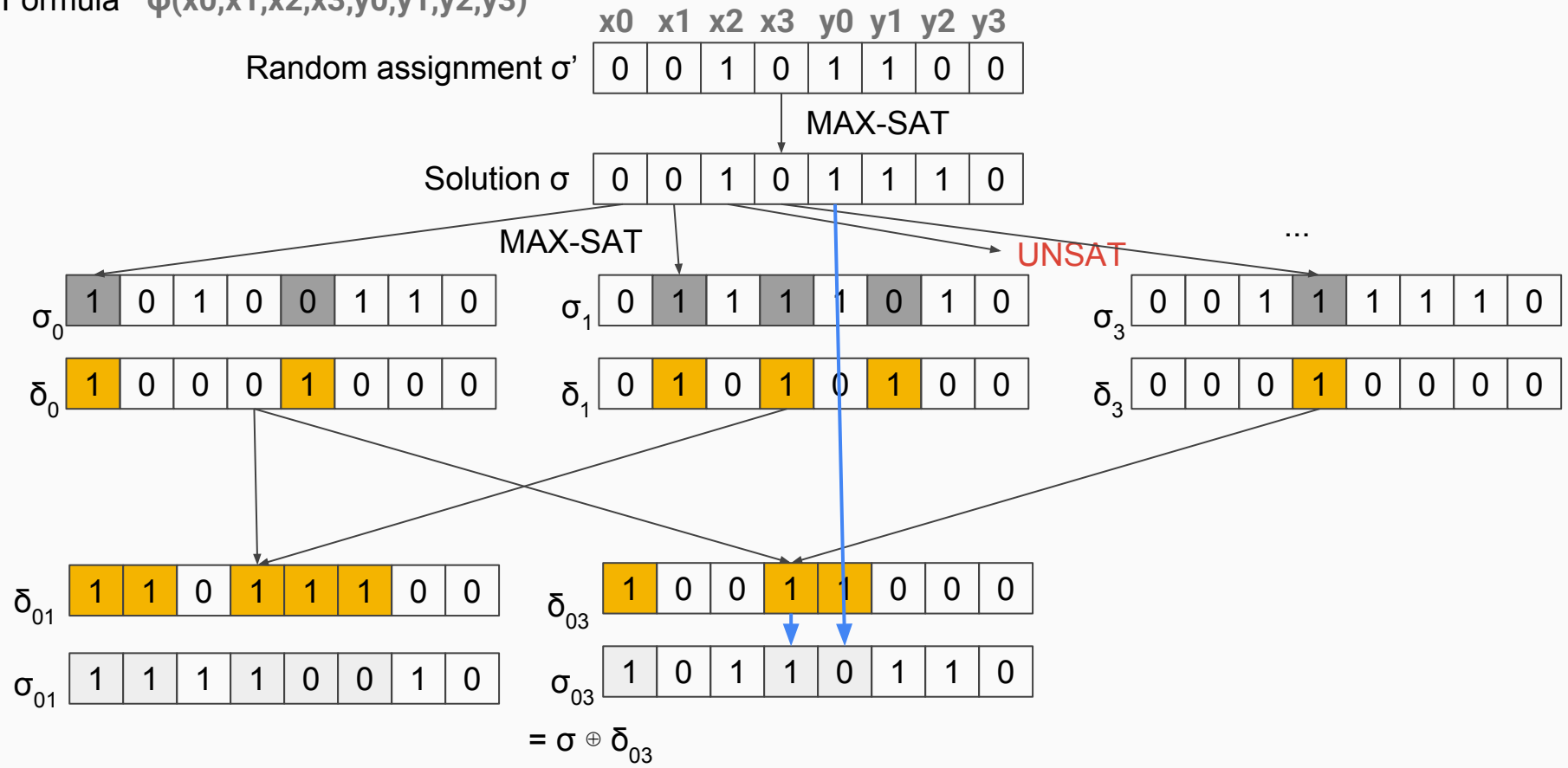
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



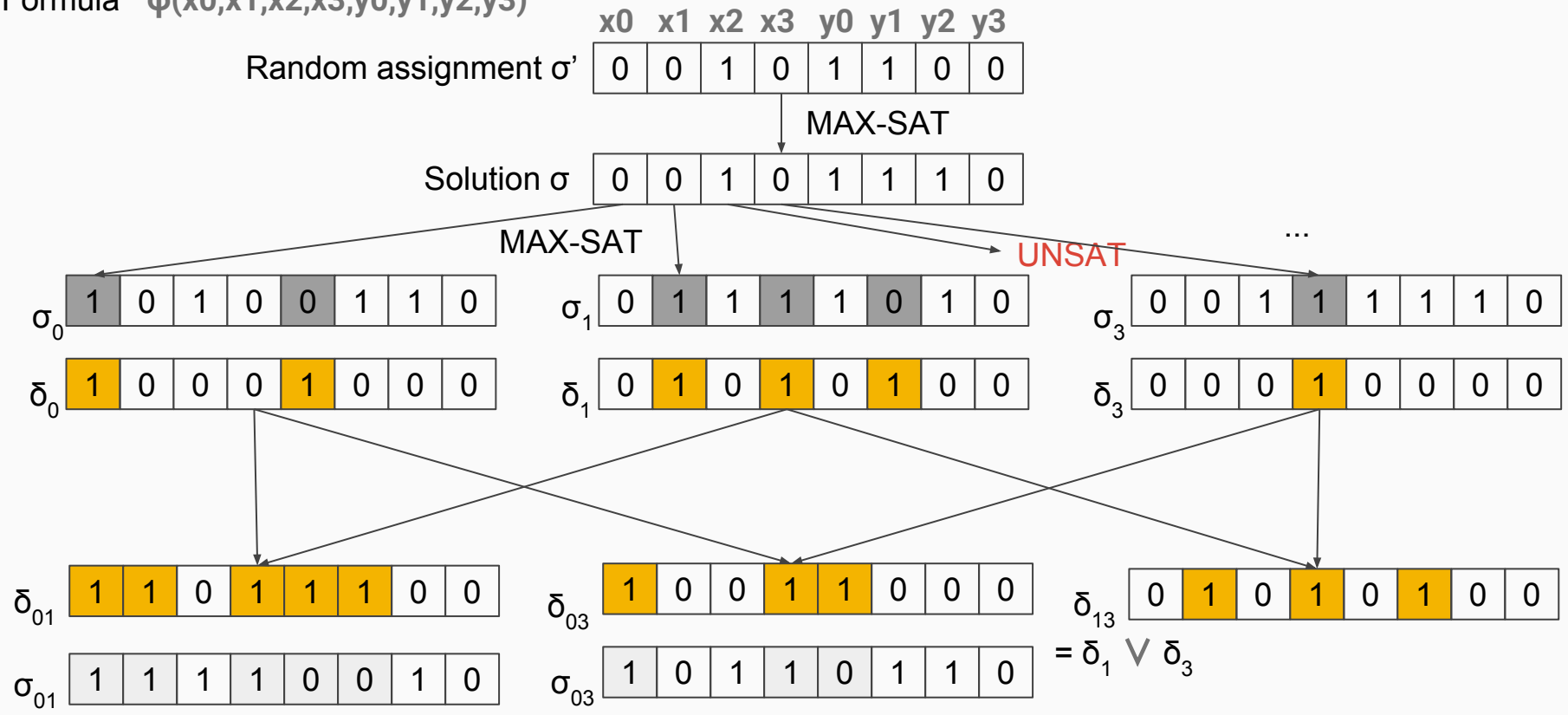
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



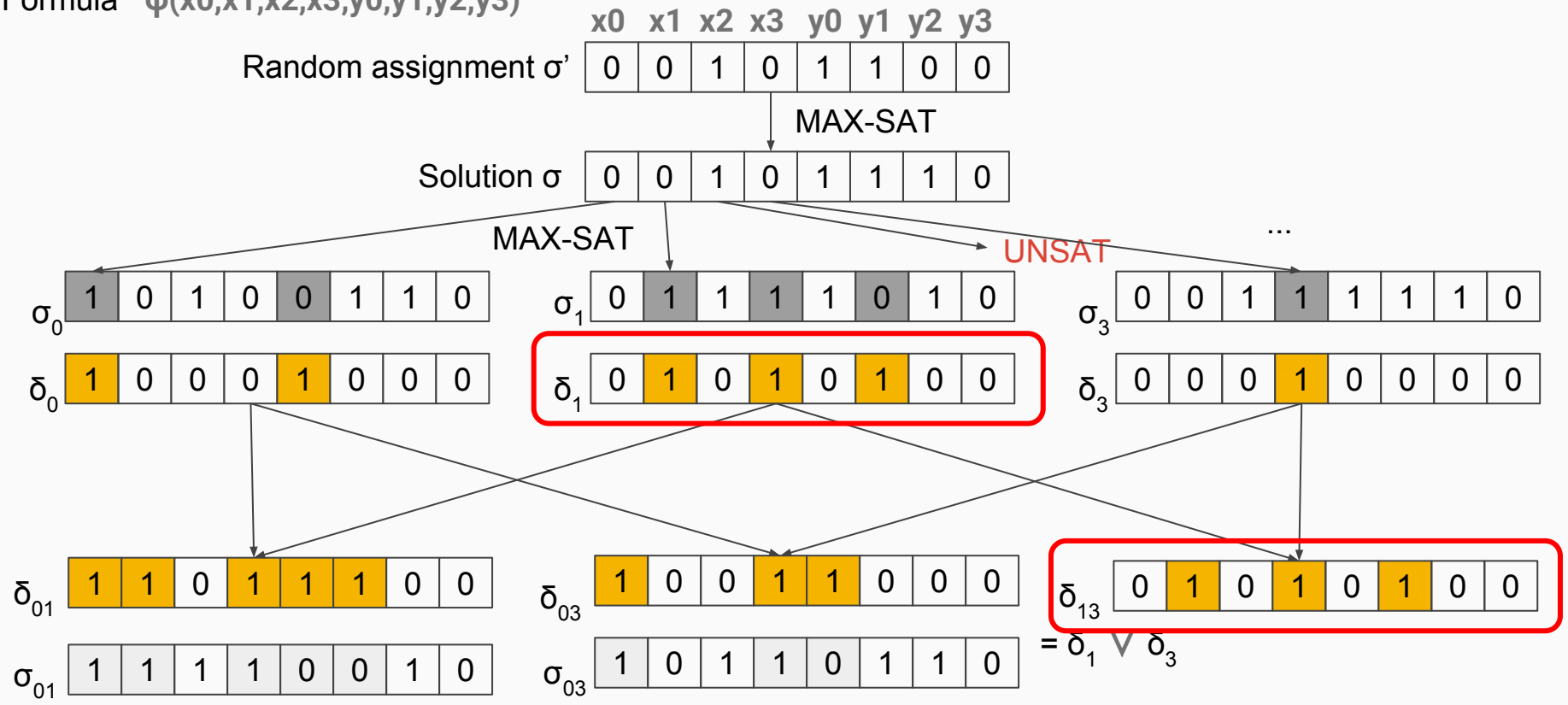
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



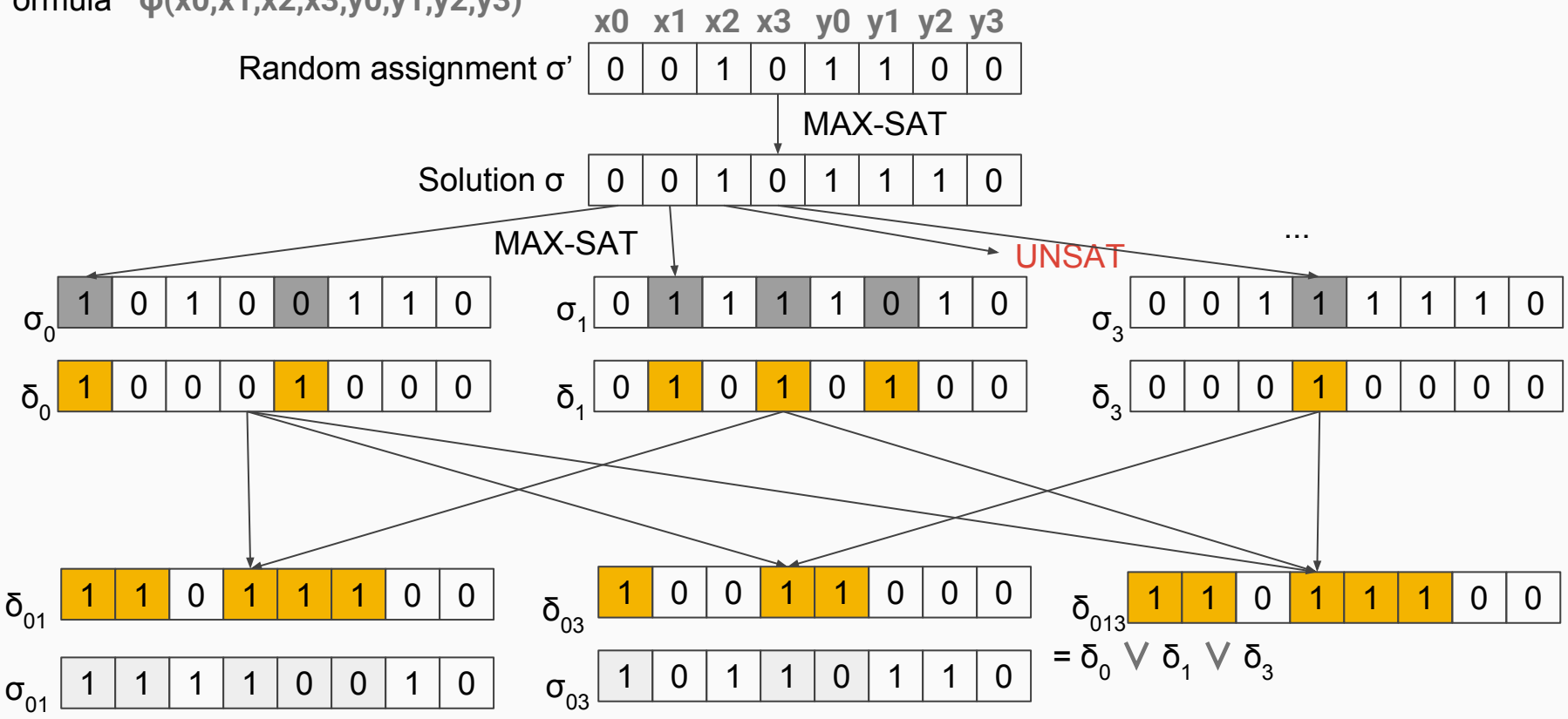
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



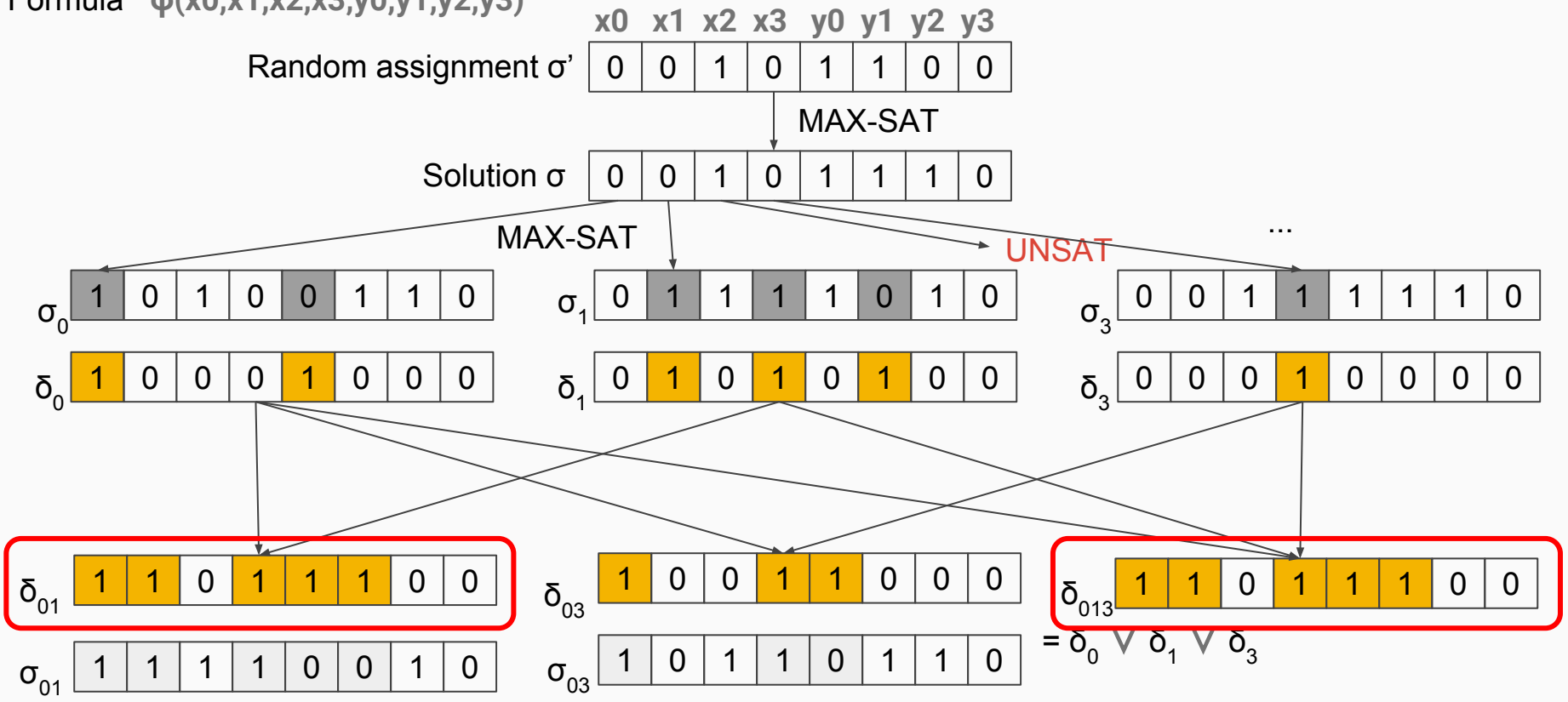
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



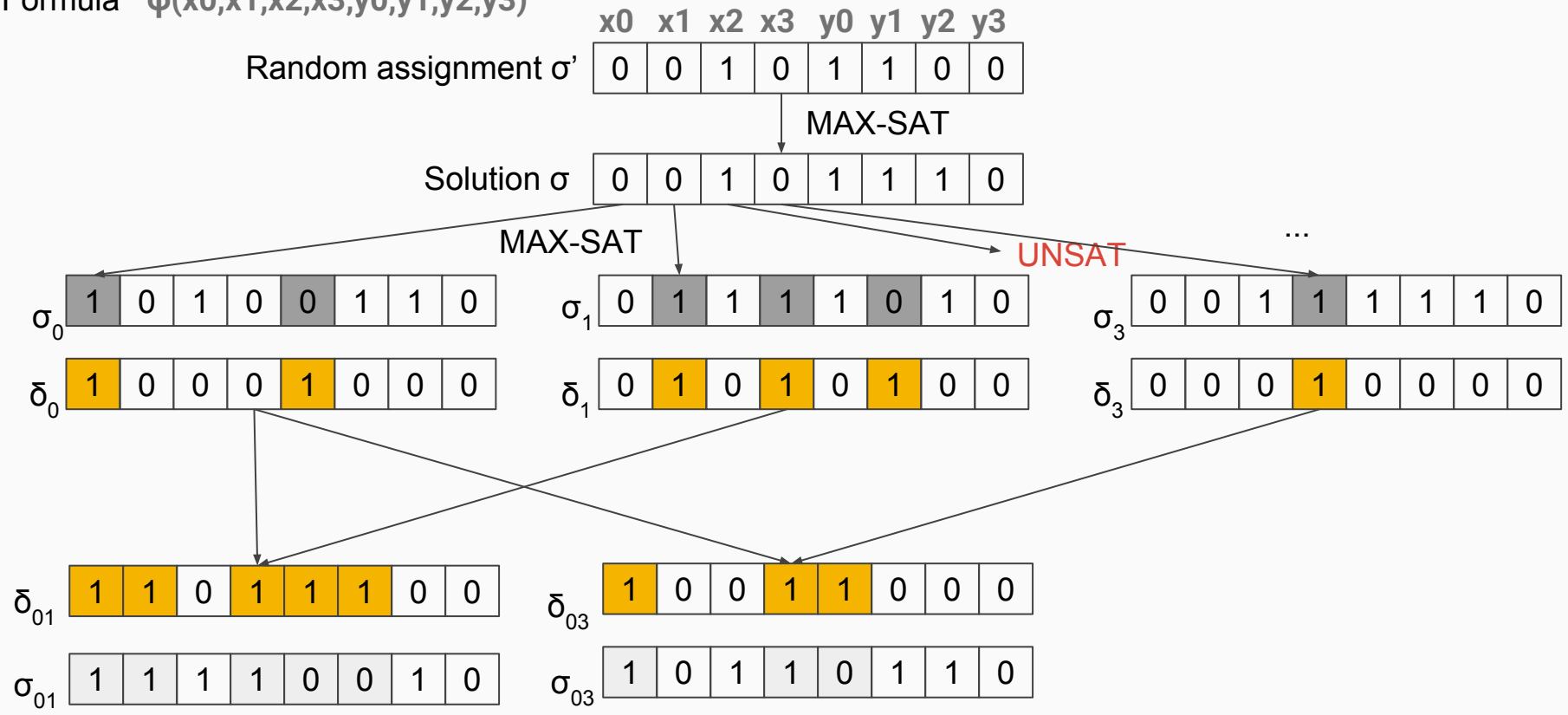
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



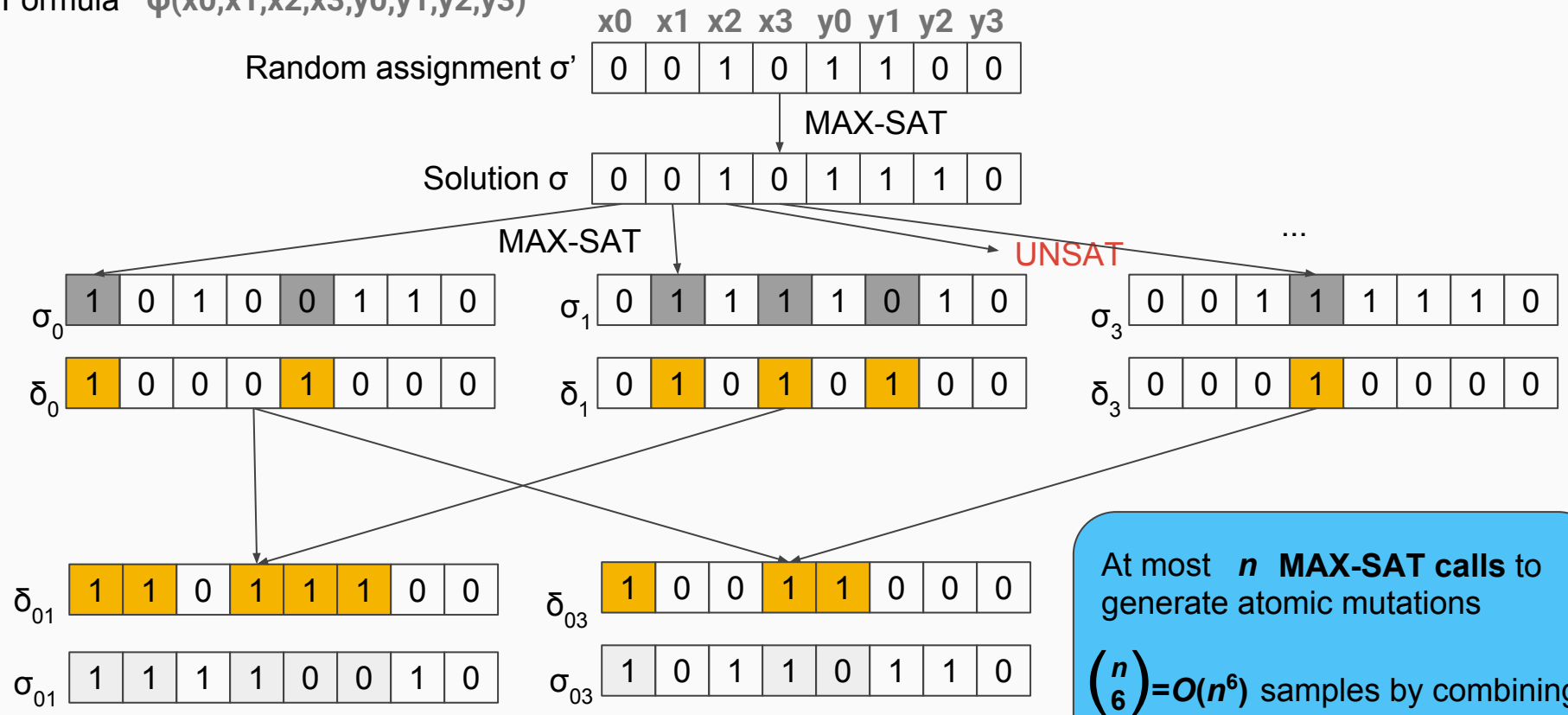
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



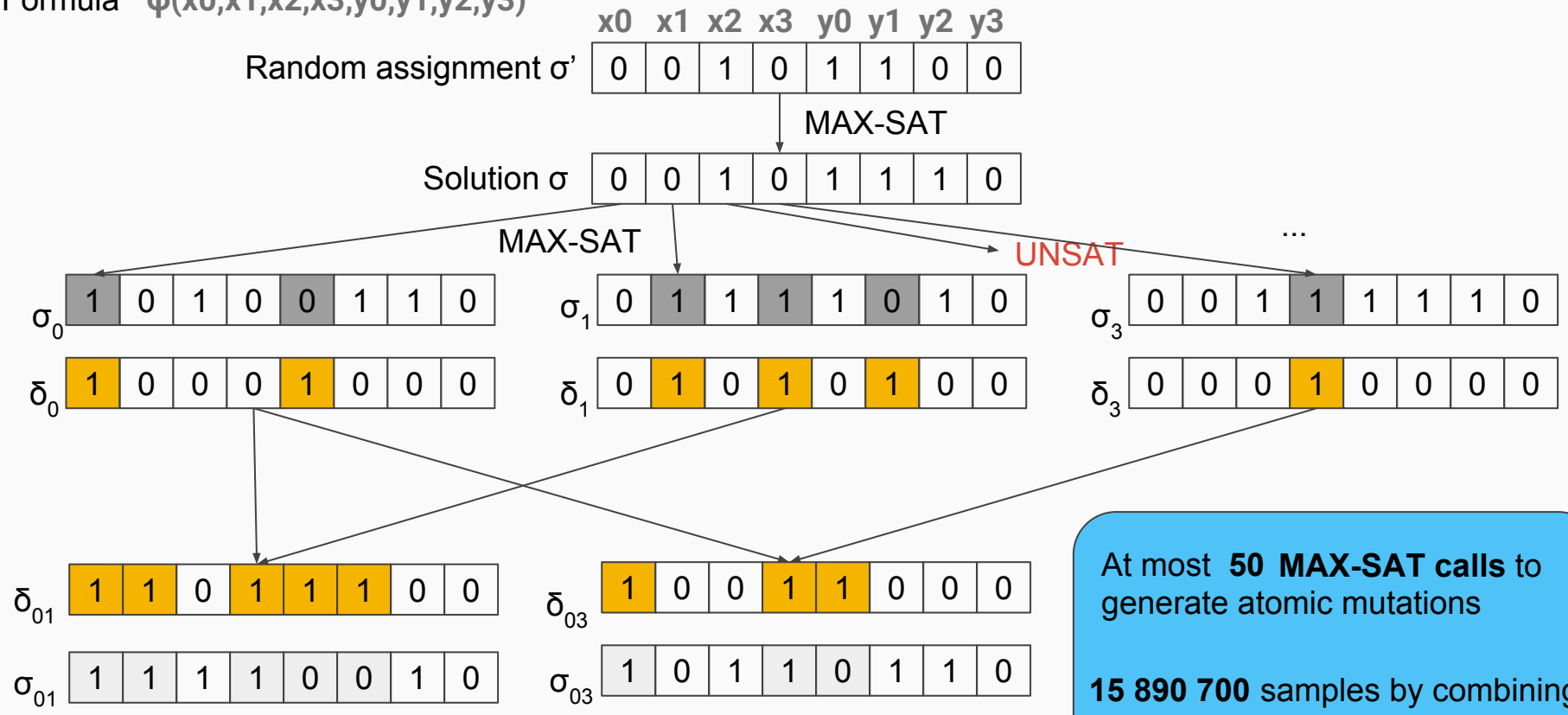
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



At most n **MAX-SAT calls** to generate atomic mutations

$\binom{n}{6} = O(n^6)$ samples by combining mutations: **NO MAX-SAT**

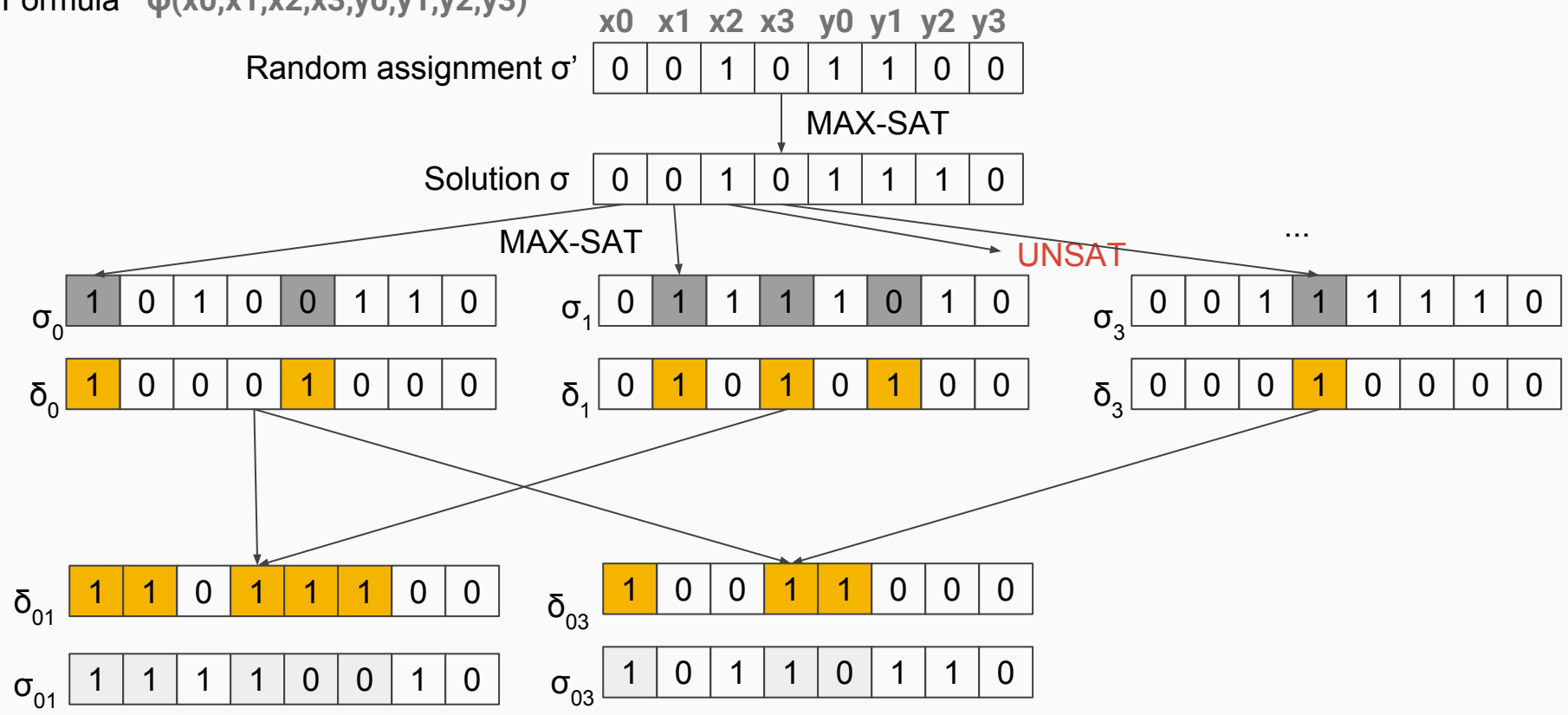
Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



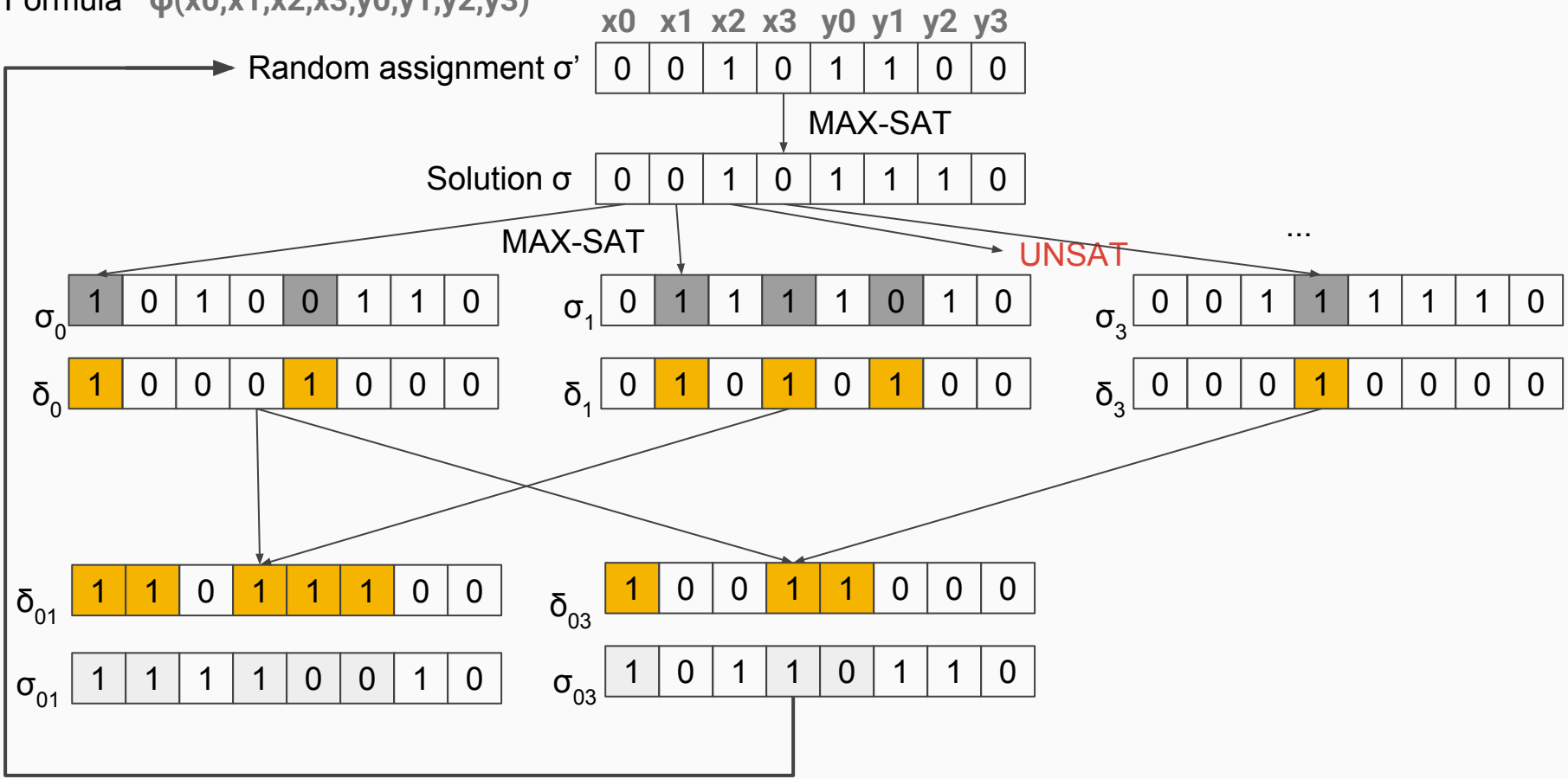
At most **50 MAX-SAT calls** to generate atomic mutations

15 890 700 samples by combining mutations: **NO MAX-SAT**

Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



Formula $\varphi(x0,x1,x2,x3,y0,y1,y2,y3)$



Implementation

- Implemented in C++ using Z3 as the constraint solver
- https://github.com/RafaelTupynamba/quicksampler

Optimizations:

- Eager generation of samples
- Independent support
- Unsatisfiable variables

Experiments

We evaluated QuickSampler on 163 industrial benchmarks.

Largest 6 Benchmarks	# Variables	# Clauses
tutorial3.sk_4_31	486193	2598178
diagStencilClean.sk_41_36	378131	2110471
karatsuba.sk_7_41	19594	82417
enqueueSeqSK.sk_10_42	16466	58515
20.sk_1_51	15475	60994
77.sk_3_44	14535	27573

Experiments

We compared QuickSampler against two state-of-the-art samplers:

- UniGen2 [1]
  - Uses universal hashing to partition the solution space and produce provably uniform samples
- SearchTreeSampler [2]
  - Generates pseudo-solutions: partial assignments that can be completed to full solutions

[1] Supratik Chakraborty, Daniel J Fremont, Kuldeep S Meel, Sanjit A Seshia, and Moshe Y Vardi. 2015. On Parallel Scalable Uniform SAT Witness Generation. In TACAS 2015.
[2] Stefano Ermon, Carla P Gomes, and Bart Selman. 2012. Uniform solution sampling using a constraint solver as an oracle. In UAI 2012.

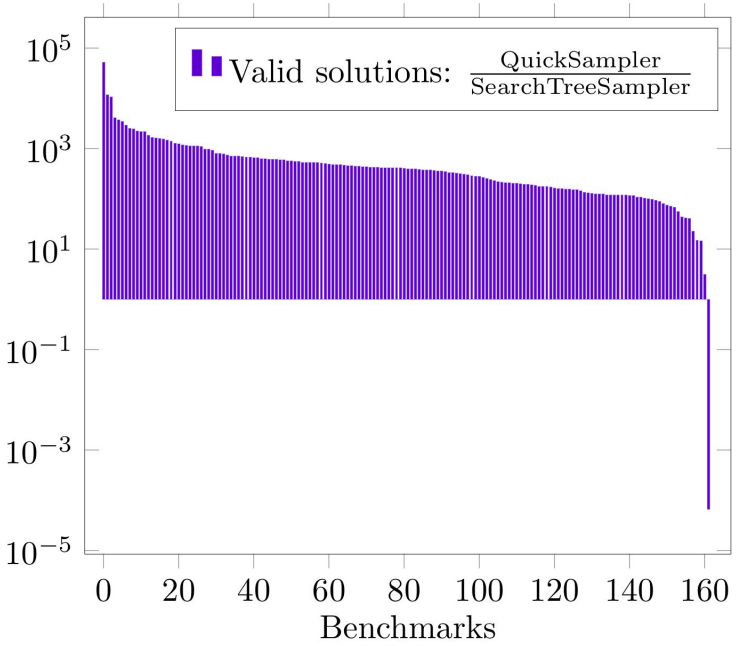
Experiments: Correctness

# of Atomic Mutations combined	Average Samples generated	% of Valid Samples
0	1	100%
1	32	100%
2	511	96%
3	5 619	93%
4	47 493	89%
5	346 367	82%
6	2 143 385	73%
Total	2 543 409	75%

- QuickSampler generates valid solutions
  - $10^{2.5\pm0.8}$ times faster than SearchTreeSampler
  - $10^{4.7\pm1.0}$ times faster than UniGen2
- QuickSampler generates *unique* valid solutions
  - $10^{2.3\pm0.7}$ times faster than SearchTreeSampler
  - $10^{4.4\pm1.1}$ times faster than UniGen2

Experiments: Speed

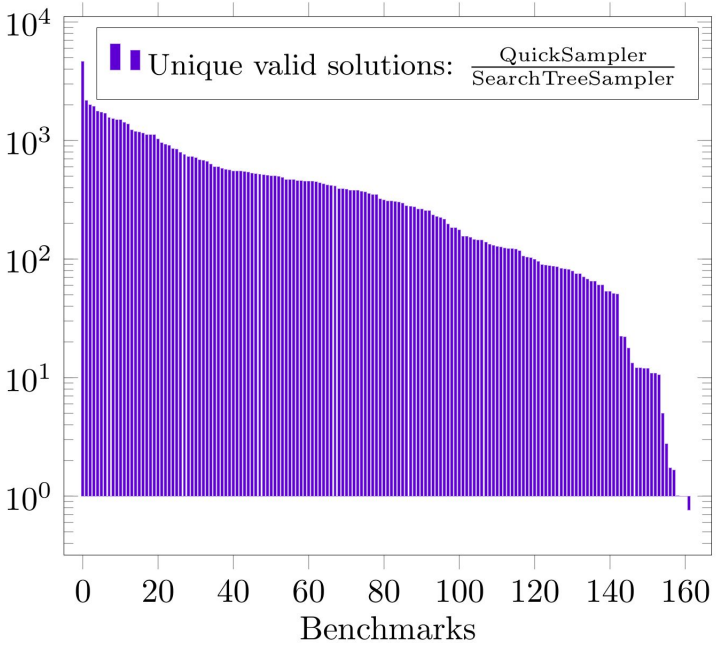
Number of valid solutions per time



Higher is better

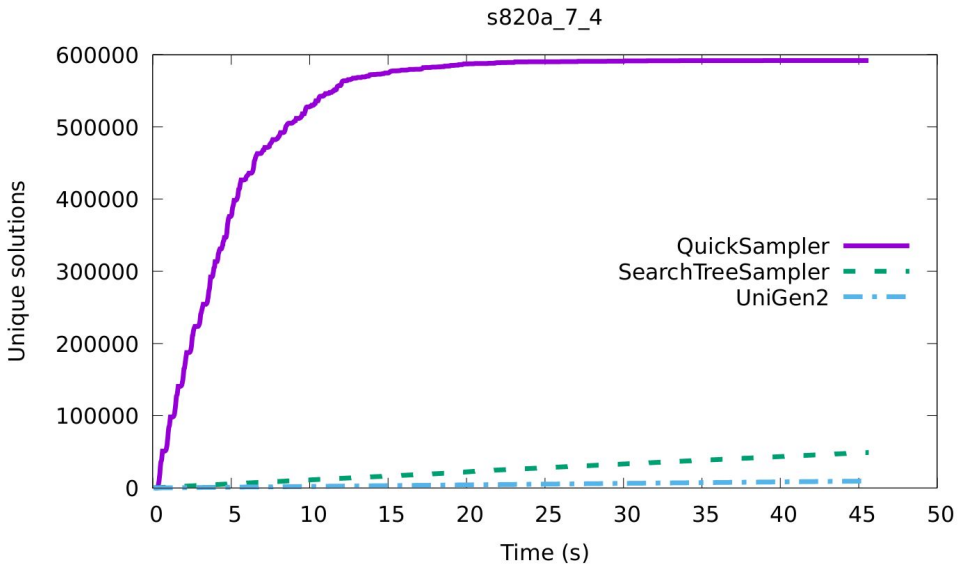
Experiments: Unique Solutions

Number of **unique** valid solutions per time



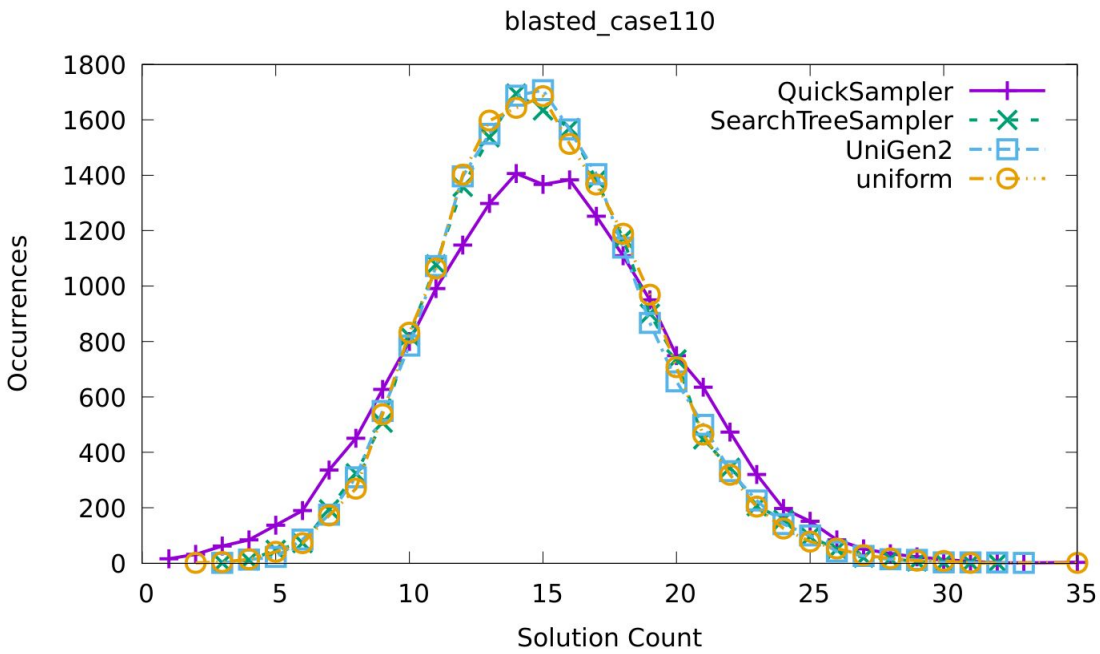
Higher is better

Experiments: Unique Solutions



Higher is better

Experiments: Uniformity



Conclusion

- Generating lots of solutions efficiently given a SAT formula
- QuickSampler can generate millions of samples with tens of solver calls
- The samples satisfy the formula with high probability and are reasonably uniform