CrystalBall: **Design a SAT Solver as you need it**

Arijit Shaw

Kuldeep S. Meel, Mate Soos

Raghav Kulkarni

Institute for Advancing Intelligence

tcg crest

School of Computing

The SAT problem

$$\mathrm{F}_1 = (a \vee b) \wedge (\neg b \vee c)$$

The SAT problem

$$\mathrm{F}_1 = \overbrace{(a \vee b)}^{c_1} \wedge \overbrace{(\neg b \vee c)}^{c_2}$$

The SAT problem

$$\mathrm{F}_1 = \overbrace{(a \vee b)}^{c_1} \wedge \overbrace{(\neg b \vee c)}^{c_2}$$

Satisfiable.
(a = ⊤, b = ⊤, c = ⊤)

The SAT problem

$$\mathrm{F}_1 = \overbrace{(a \vee b)}^{c_1} \wedge \overbrace{(\neg b \vee c)}^{c_2}$$

Satisfiable.
(a = ⊤, b = ⊤, c = ⊤)

$$\mathrm{F}_2 = (a \vee b) \wedge (\neg b \vee c) \wedge (a) \wedge (\neg c)$$

The SAT problem

$$\mathrm{F}_1 = \overbrace{(a \vee b)}^{c_1} \wedge \overbrace{(\neg b \vee c)}^{c_2}$$

Satisfiable.
(a = ⊤, b = ⊤, c = ⊤)

$$\mathrm{F}_2 = (a \vee b) \wedge (\neg b \vee c) \wedge (a) \wedge (\neg c)$$

Unsatisfiable.

The SAT problem

$$\mathrm{F}_1 = \overbrace{(a \vee b)}^{c_1} \wedge \overbrace{(\neg b \vee c)}^{c_2}$$

Satisfiable.
(a = ⊤, b = ⊤, c = ⊤)

$$\mathrm{F}_2 = (a \vee b) \wedge (\neg b \vee c) \wedge (a) \wedge (\neg c)$$

Unsatisfiable.

A lot can be encoded:

The SAT problem

$$F_1 = \overbrace{(a \vee b)}^{c_1} \wedge \overbrace{(\neg b \vee c)}^{c_2}$$

Satisfiable.
(a = ⊤, b = ⊤, c = ⊤)

$$F_2 = (a \vee b) \wedge (\neg b \vee c) \wedge (a) \wedge (\neg c)$$

Unsatisfiable.

A lot can be encoded:

- Cryptanalysis

The SAT problem

$$\mathrm{F}_1 = \overbrace{(a \vee b)}^{c_1} \wedge \overbrace{(\neg b \vee c)}^{c_2}$$

Satisfiable.
(a = ⊤, b = ⊤, c = ⊤)

$$\mathrm{F}_2 = (a \vee b) \wedge (\neg b \vee c) \wedge (a) \wedge (\neg c)$$

Unsatisfiable.

A lot can be encoded:

- Cryptanalysis
- Optimize the S-Box

The SAT problem

$$F_1 = \overbrace{(a \vee b)}^{c_1} \wedge \overbrace{(\neg b \vee c)}^{c_2}$$

Satisfiable.
(a = ⊤, b = ⊤, c = ⊤)

$$F_2 = (a \vee b) \wedge (\neg b \vee c) \wedge (a) \wedge (\neg c)$$

Unsatisfiable.

A lot can be encoded:

- Cryptanalysis
- Optimize the S-Box
- Get the optimized decision tree

The SAT problem

$$\mathrm{F}_1 = \overbrace{(a \vee b)}^{c_1} \wedge \overbrace{(\neg b \vee c)}^{c_2}$$

Satisfiable.
(a = ⊤, b = ⊤, c = ⊤)

$$\mathrm{F}_2 = (a \vee b) \wedge (\neg b \vee c) \wedge (a) \wedge (\neg c)$$

Unsatisfiable.

A lot can be encoded:

- Cryptanalysis
- Optimize the S-Box
- Get the optimized decision tree
- Prove theorem

The SAT problem

$$F_1 = \overbrace{(a \vee b)}^{c_1} \wedge \overbrace{(\neg b \vee c)}^{c_2}$$

Satisfiable.
(a = ⊤, b = ⊤, c = ⊤)

$$F_2 = (a \vee b) \wedge (\neg b \vee c) \wedge (a) \wedge (\neg c)$$

Unsatisfiable.

A lot can be encoded:

- Cryptanalysis
- Optimize the S-Box
- Get the optimized decision tree
- Prove theorem
- Problems in chessboard

The SAT problem

$$F_1 = \overbrace{(a \vee b)}^{c_1} \wedge \overbrace{(\neg b \vee c)}^{c_2}$$

Satisfiable.
(a = ⊤, b = ⊤, c = ⊤)

$$F_2 = (a \vee b) \wedge (\neg b \vee c) \wedge (a) \wedge (\neg c)$$

Unsatisfiable.

A lot can be encoded:

- Cryptanalysis
- Optimize the S-Box
- Get the optimized decision tree
- Prove theorem
- Problems in chessboard

SAT is NP-Complete

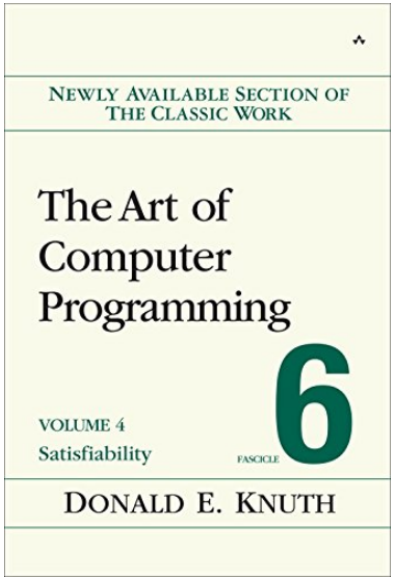
SAT is NP-Complete

In 80's : Reduce SAT to a problem

– To show it's difficuilty

21st century : Reduce a problem to SAT

– To solve it in practice



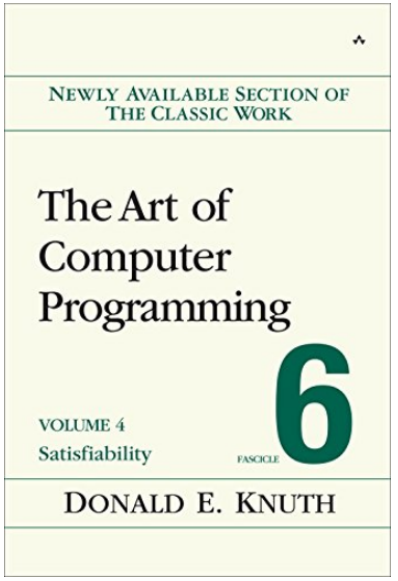
SAT is NP-Complete

In 80's : Reduce SAT to a problem

– To show it's difficuilty

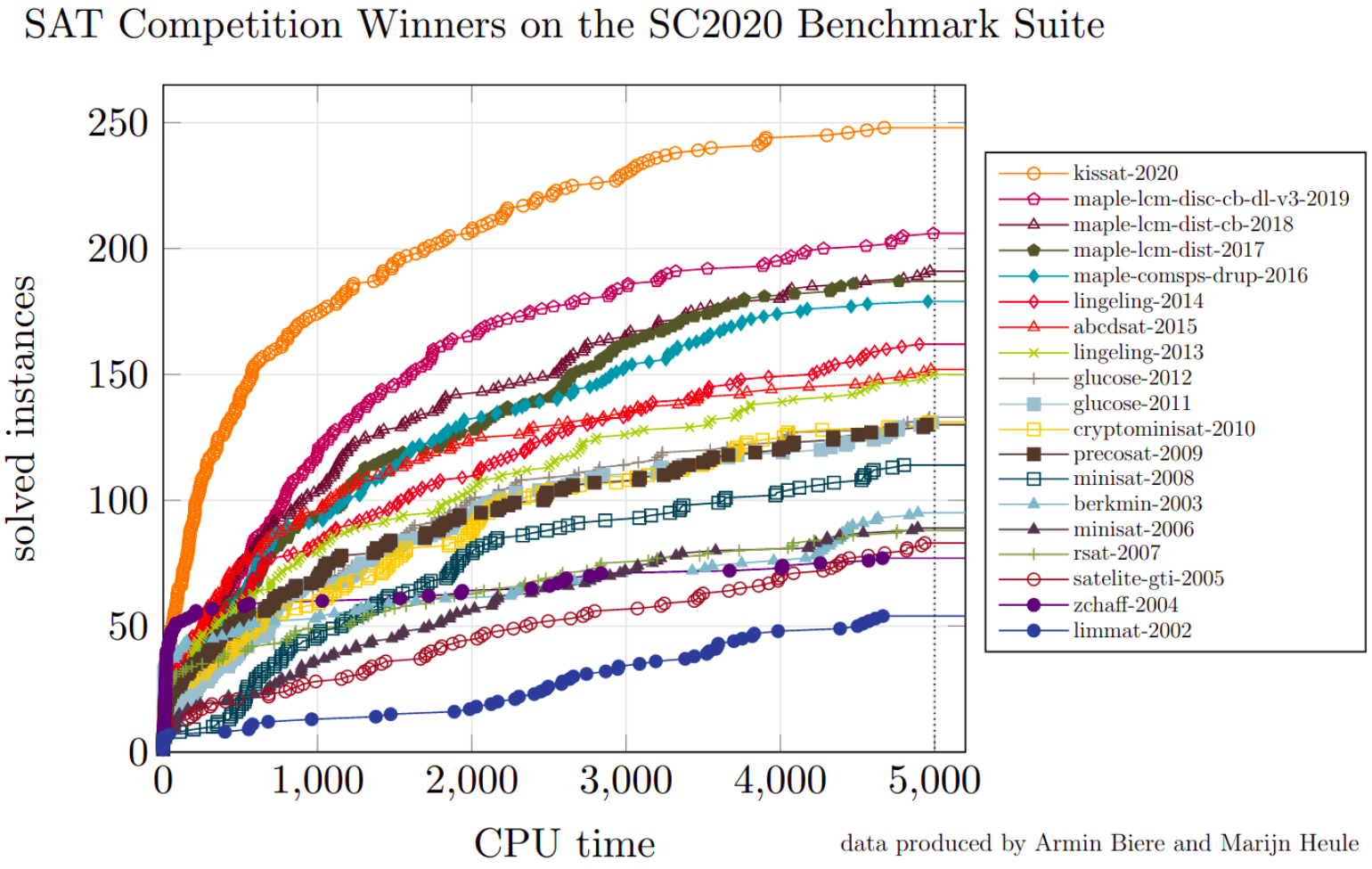
21st century : Reduce a problem to SAT

– To solve it in practice



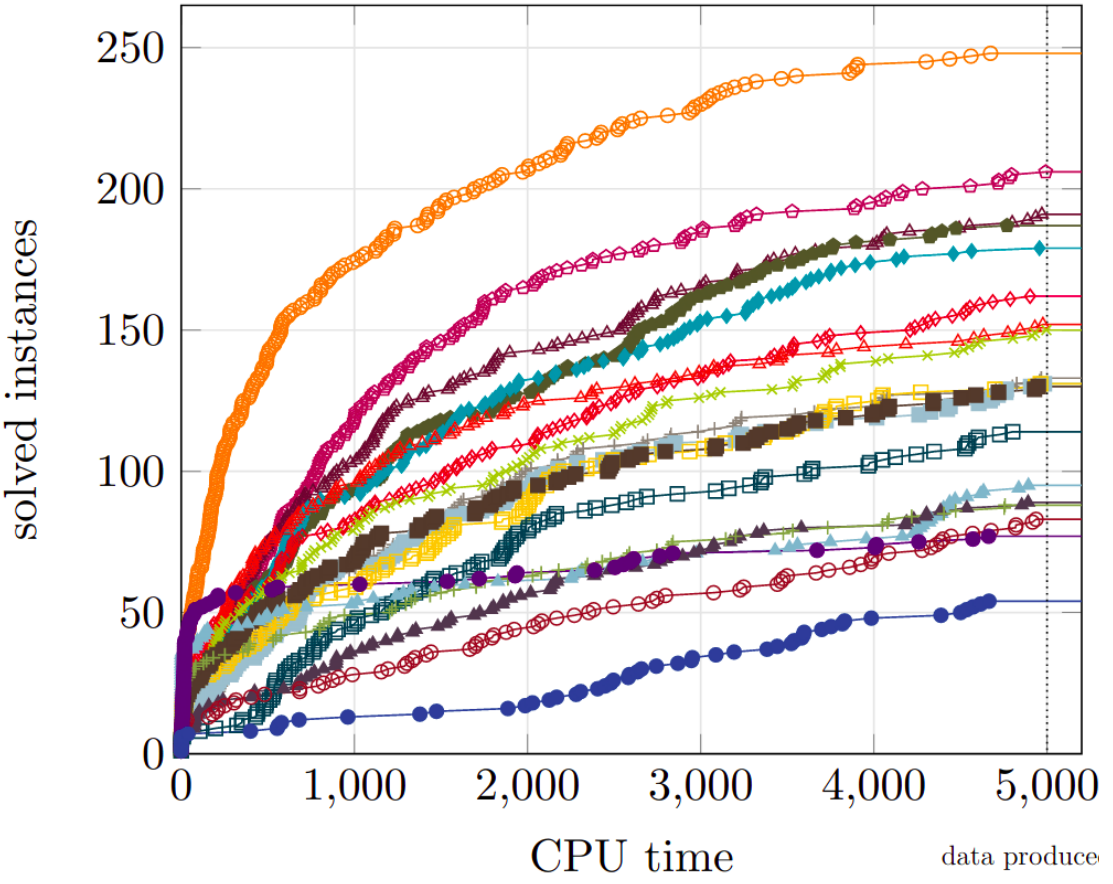
Industrial SAT solvers : Software/Hardware Verification, cryptography, AI planning, genome rearrangement, constrained scheduling...

SAT Revolution



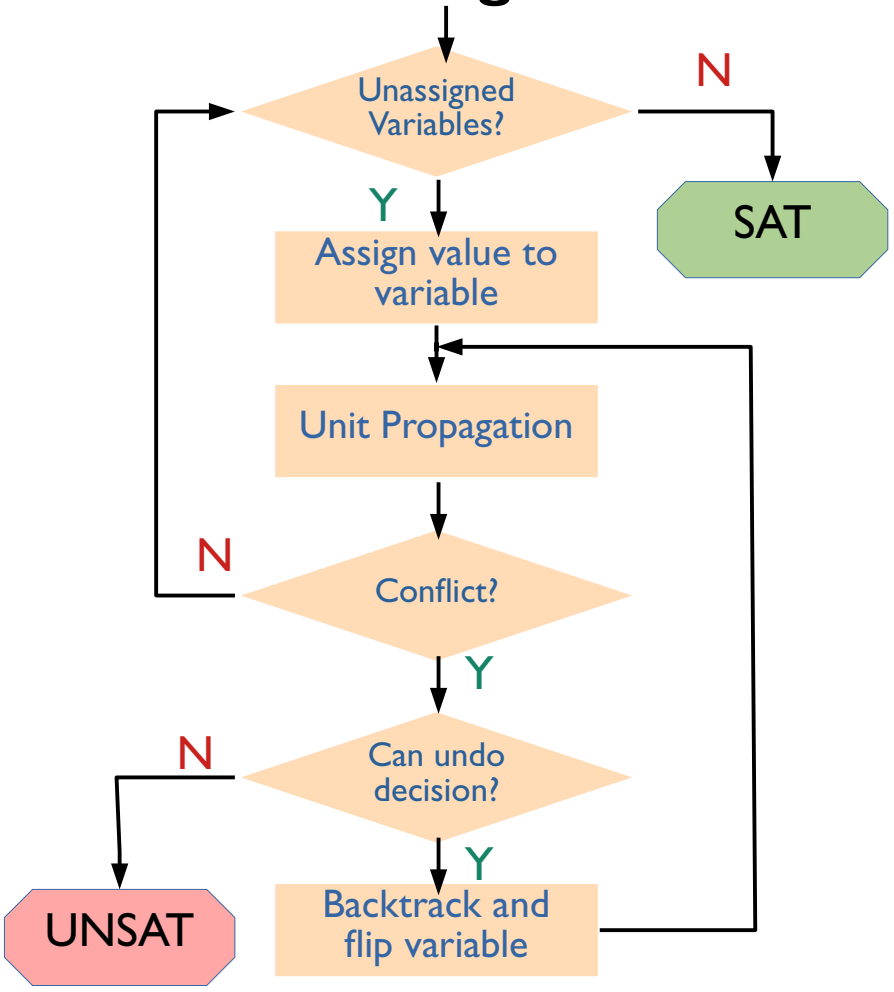
The Price of Success

- SAT is NP-complete yet solvers tend to solve problems involving millions of variables
- The solvers of today are very complex
- We understand very little why SAT solvers work!



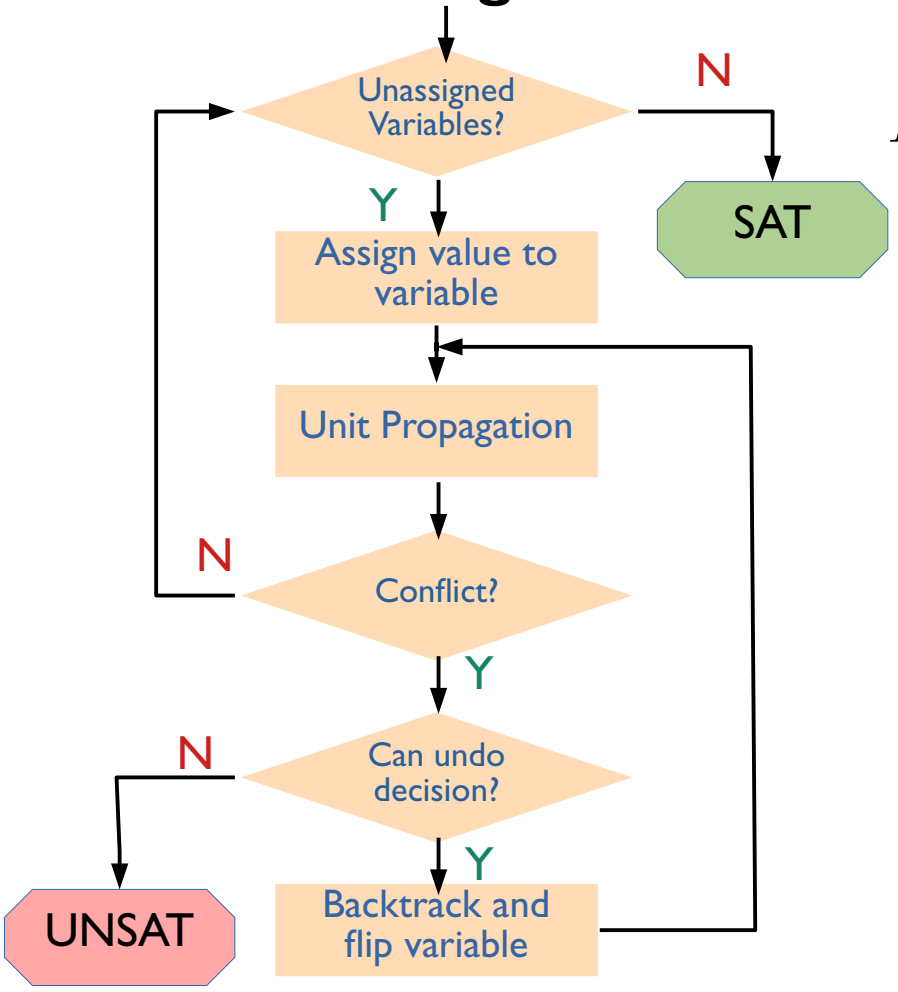
The DPLL Algorithm

[DL60, DLL62]



The DPLL Algorithm

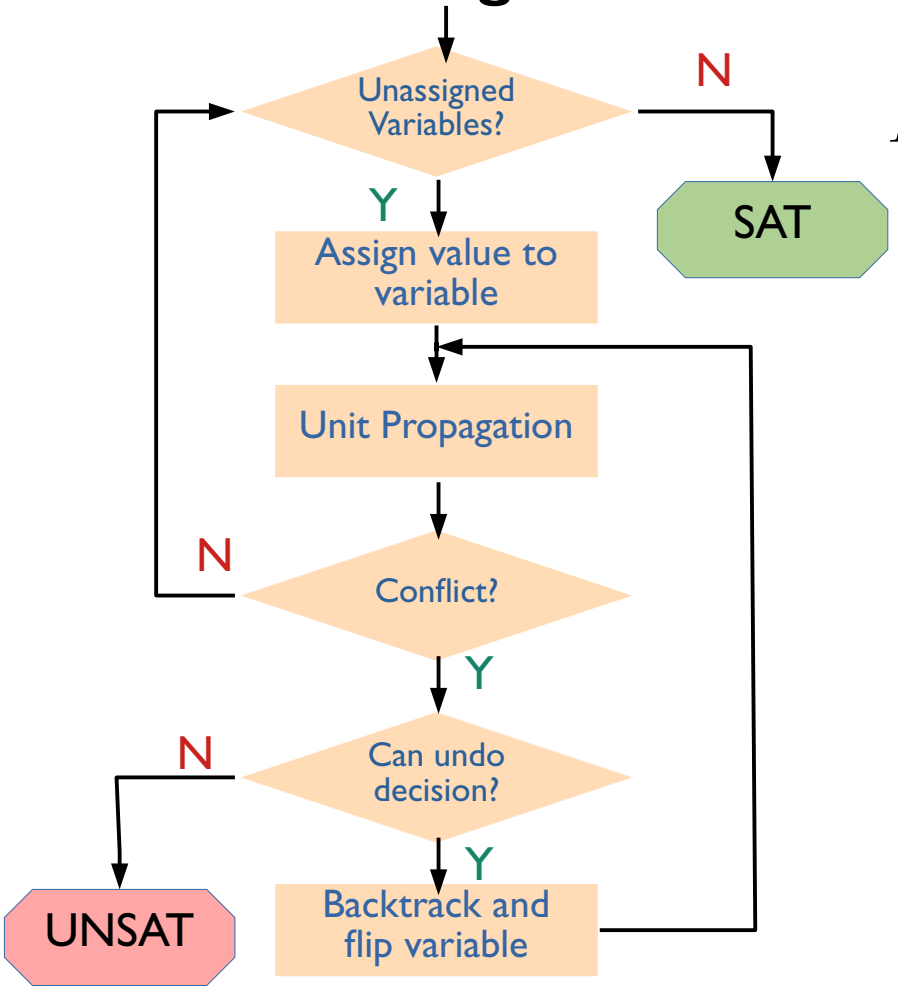
[DL60, DLL62]



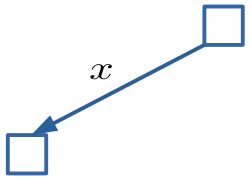
$$F = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge \left(a \vee \bar{b}\right) \wedge \left(\bar{a} \vee \bar{b}\right)$$

The DPLL Algorithm

[DL60, DLL62]



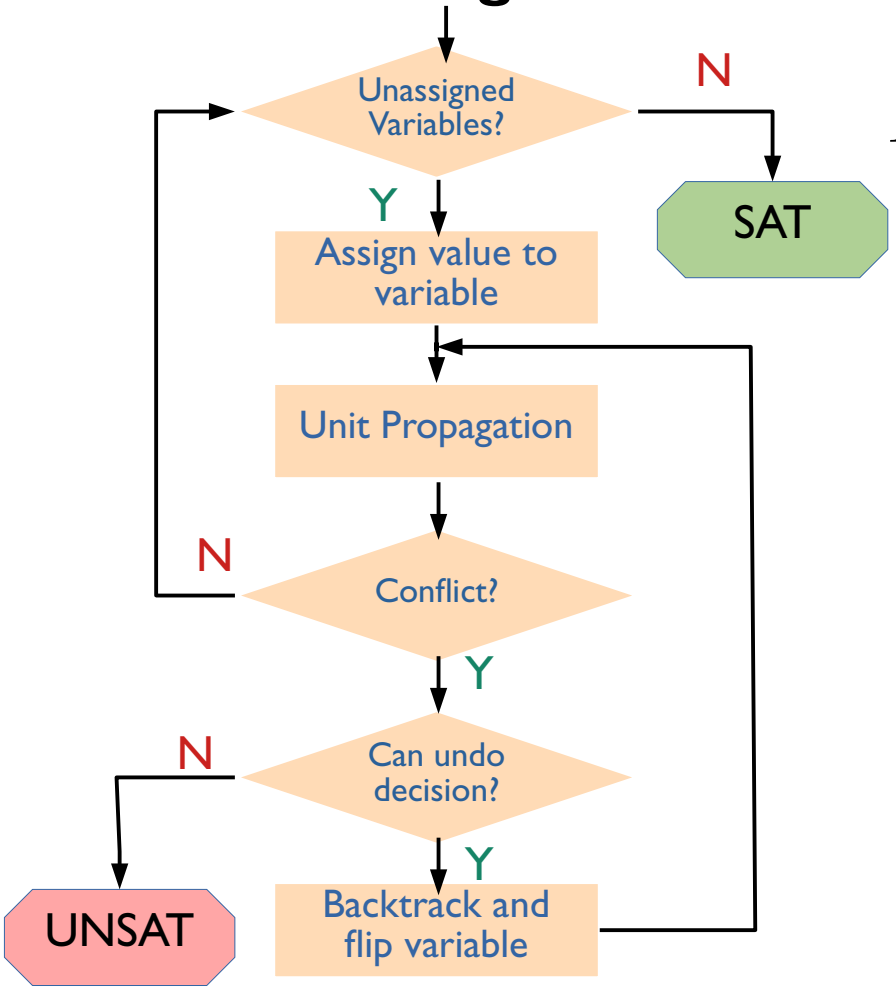
$$F = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$



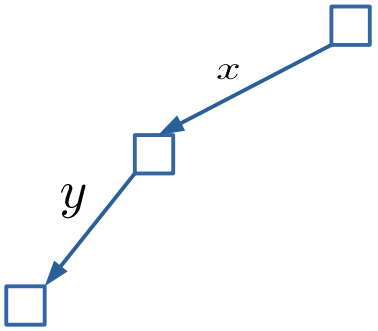
Level	Dec.	Unit Prop.
0		
1	x	
2	y	
3	a	

The DPLL Algorithm

[DL60, DLL62]



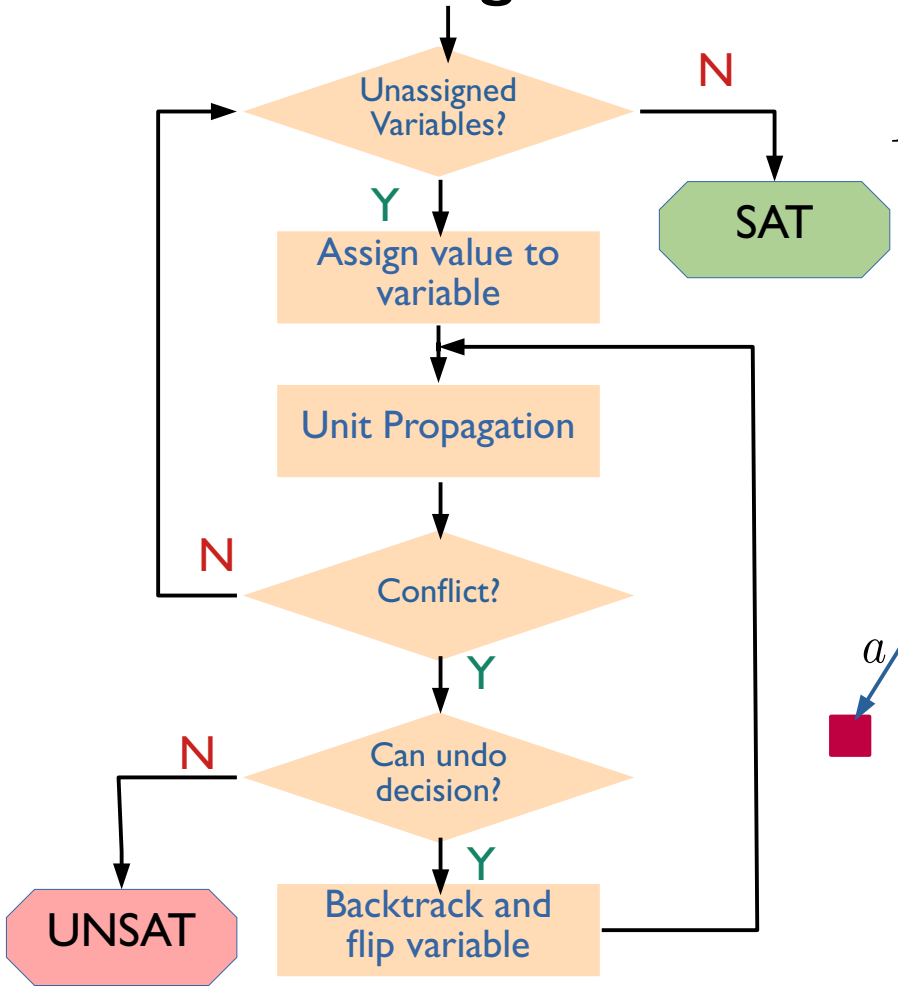
$$F = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$



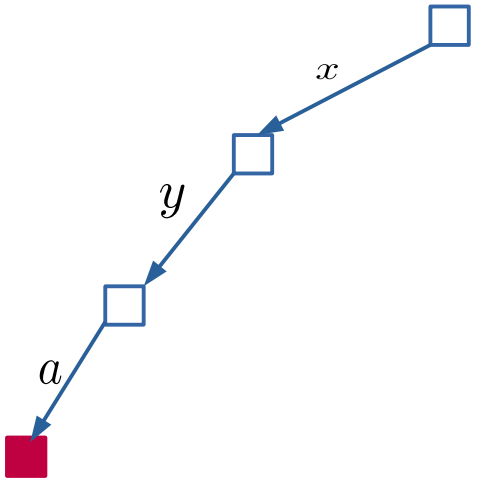
Level	Dec.	Unit Prop.
0		
1	x	
2	y	
3	a	

The DPLL Algorithm

[DL60, DLL62]



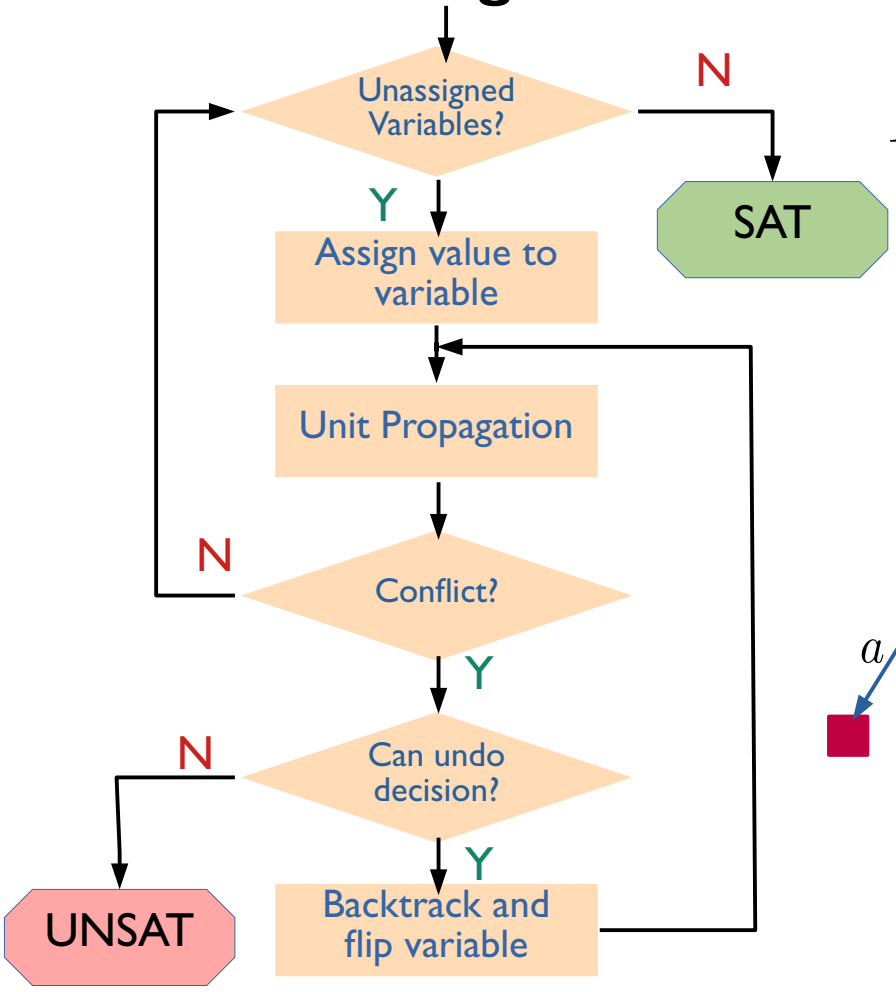
$$F = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge \left(a \vee \bar{b}\right) \wedge \left(\bar{a} \vee \bar{b}\right)$$



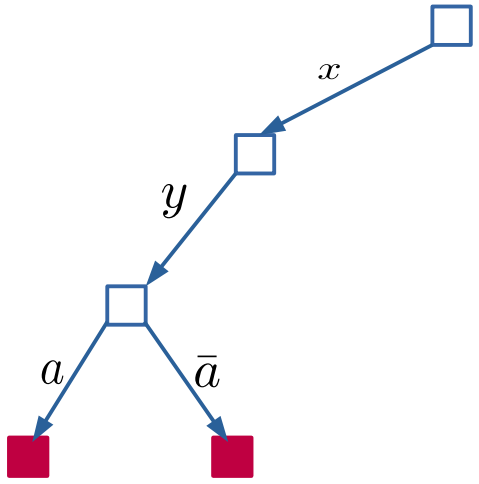
Level	Dec.	Unit Prop.
0		
1	x	
2	y	
3	a	→ b → ⊥

The DPLL Algorithm

[DL60, DLL62]



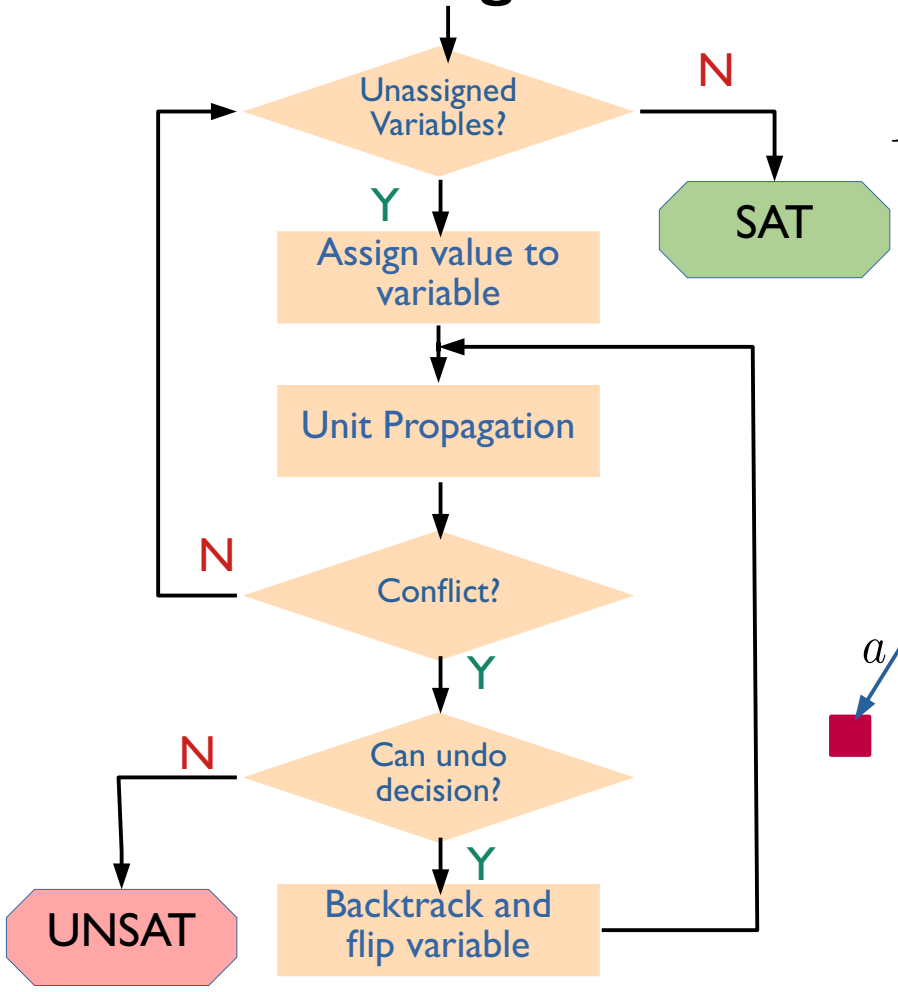
$$F = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge \left(a \vee \bar{b}\right) \wedge \left(\bar{a} \vee \bar{b}\right)$$



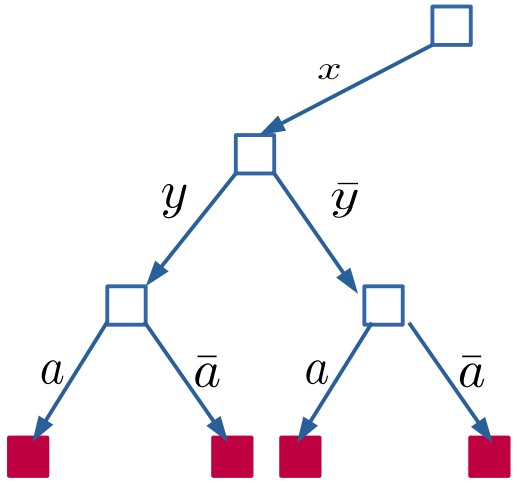
Level	Dec.	Unit Prop.
0		
1	x	
2	y	
3	a	b → $\perp$

The DPLL Algorithm

[DL60, DLL62]



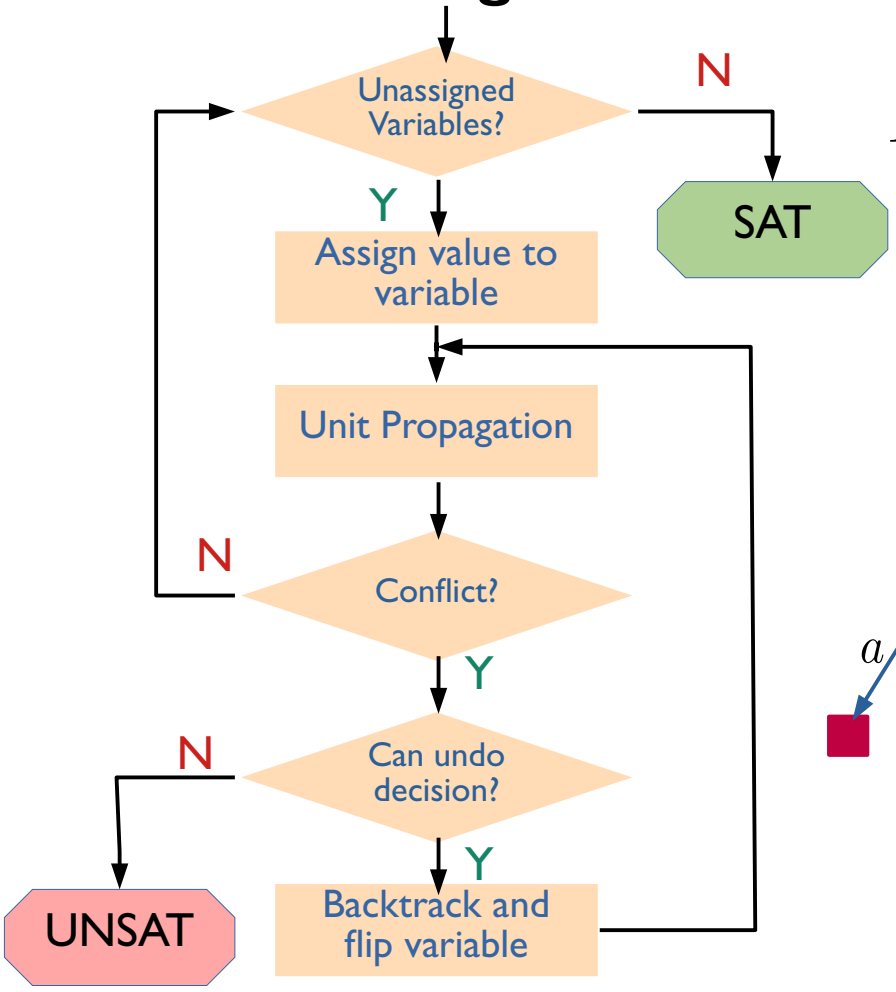
$$F = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge \left(a \vee \bar{b}\right) \wedge \left(\bar{a} \vee \bar{b}\right)$$



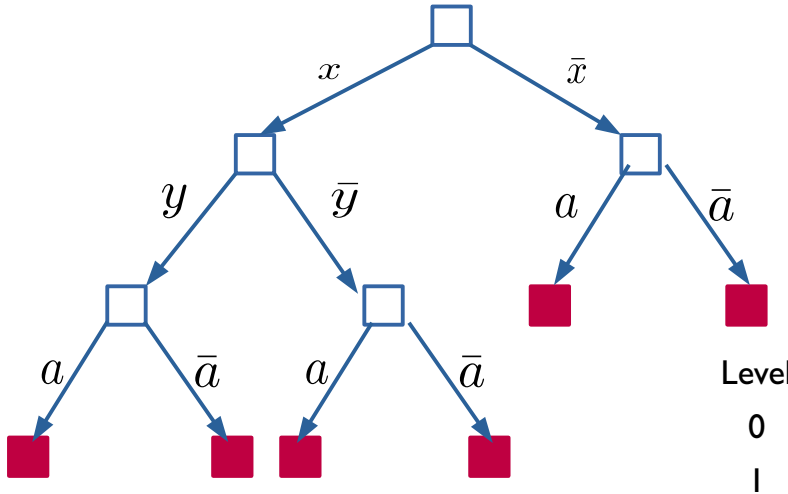
Level	Dec.	Unit Prop.
0		
1	x	
2	y	
3	a	→ b → $\bot$

The DPLL Algorithm

[DL60, DLL62]



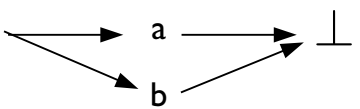
$$F = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$



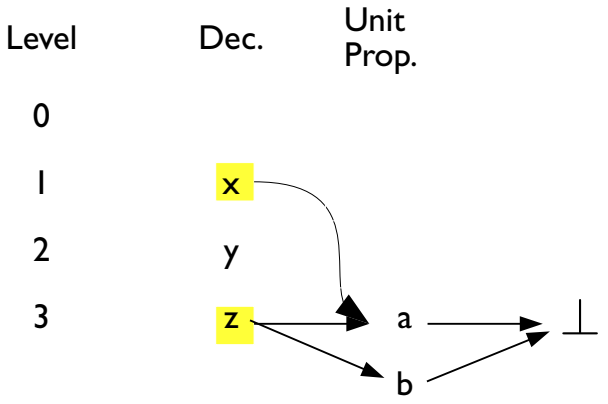
Level	Dec.	Unit Prop.
0		
1	x	
2	y	
3	a	b → ⊥

$$\left(\bar{a} \vee \bar{b}\right) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$$

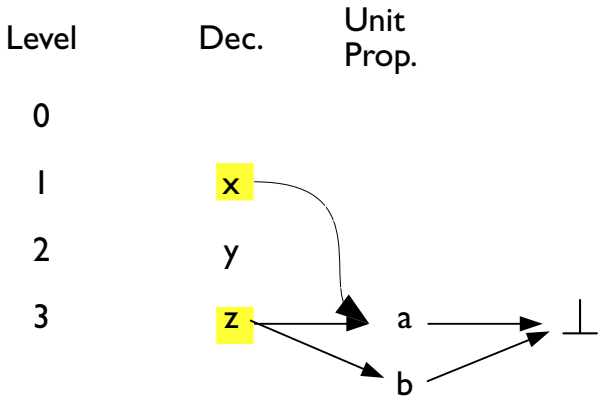
Level	Dec.	Unit Prop.
0		
1	x	
2	y	
3	z	a, b, $\bot$



$$\left(\bar{a} \vee \bar{b}\right) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$$



$$(\bar{a} \vee \bar{b}) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$$



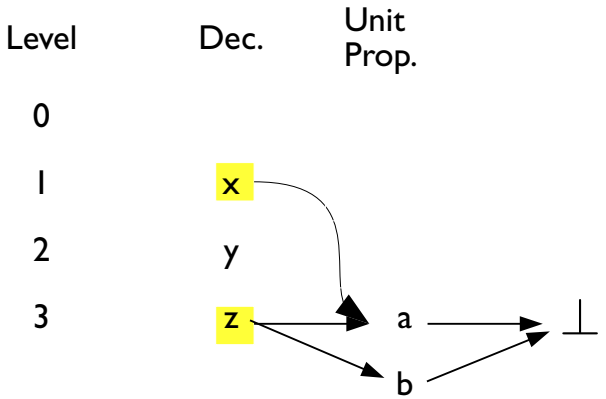
Analyze Conflict

[MSS96]

- Create a new clause $(\bar{x} \vee \bar{z})$

Conflict Driven Clause Learning (CDCL)

$$(\bar{a} \vee \bar{b}) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$$



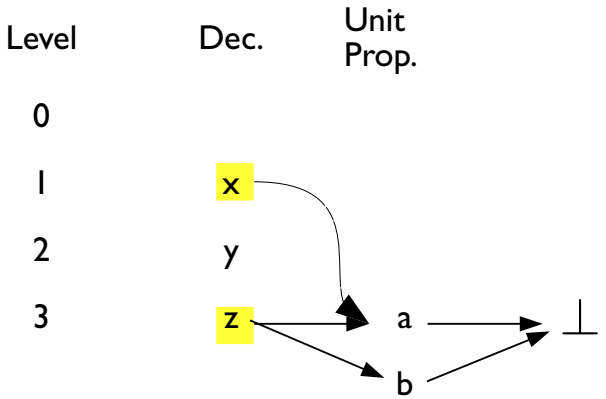
Analyze Conflict

- Create a new clause $(\bar{x} \vee \bar{z})$

[MSS96]

Conflict Driven Clause Learning (CDCL)

$$(\bar{a} \vee \bar{b}) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$$



Clause learning can be related with resolution

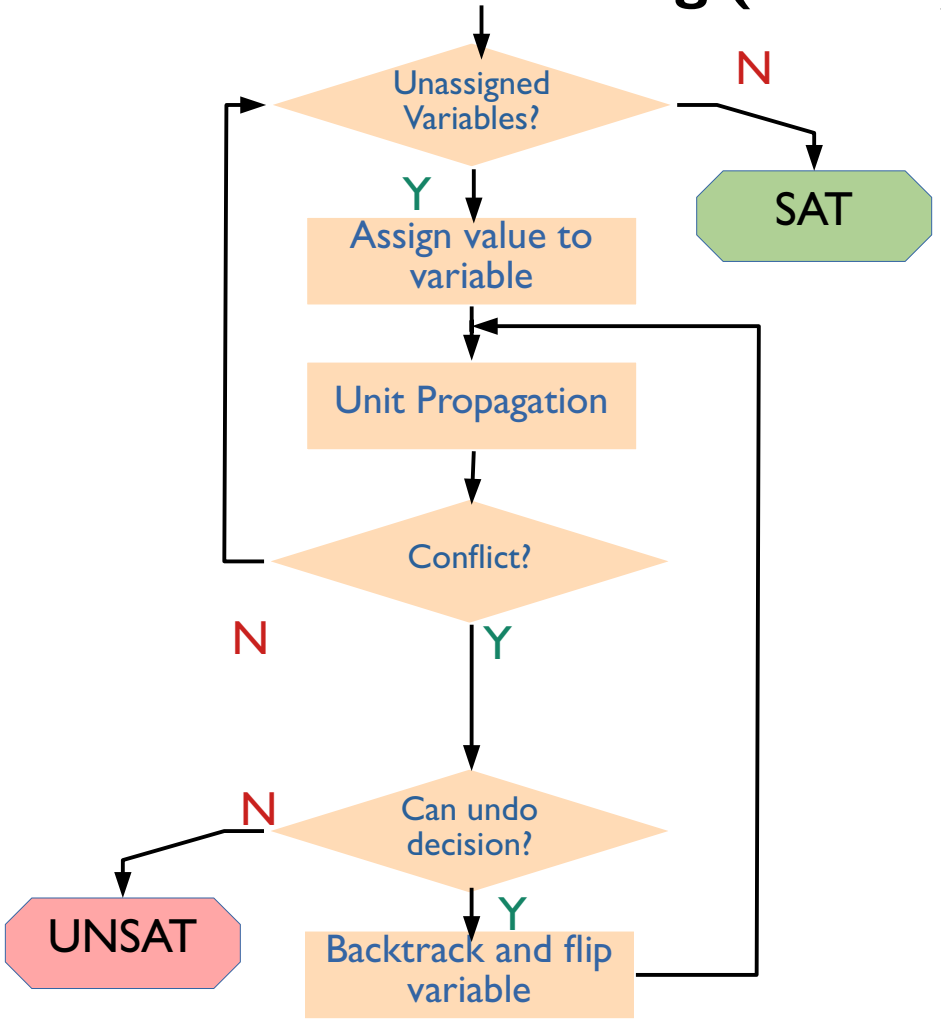
$$\frac{(\bar{a} \vee \bar{b}) \qquad (\bar{z} \vee b) \qquad (\bar{x} \vee \bar{z} \vee a)}{(\bar{x} \vee \bar{z})}$$

[MSS96]

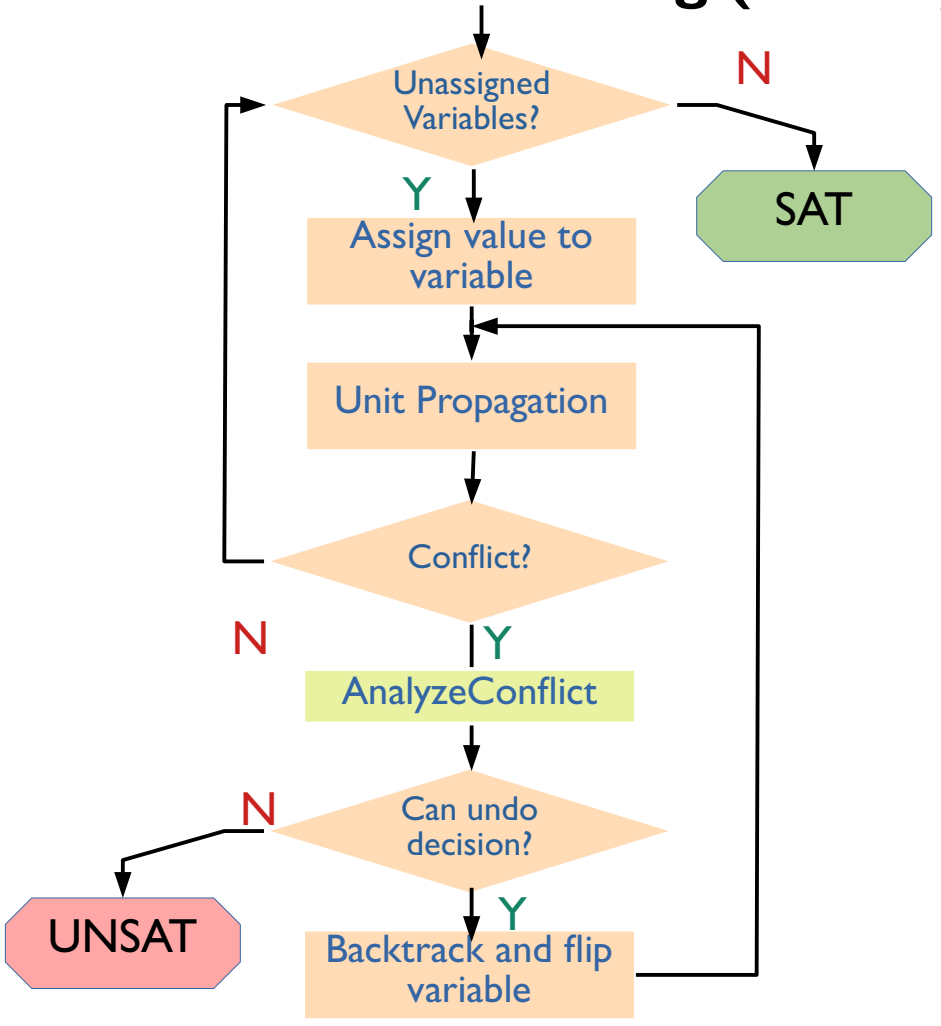
Analyze Conflict

- Create a new clause $(\bar{x} \vee \bar{z})$

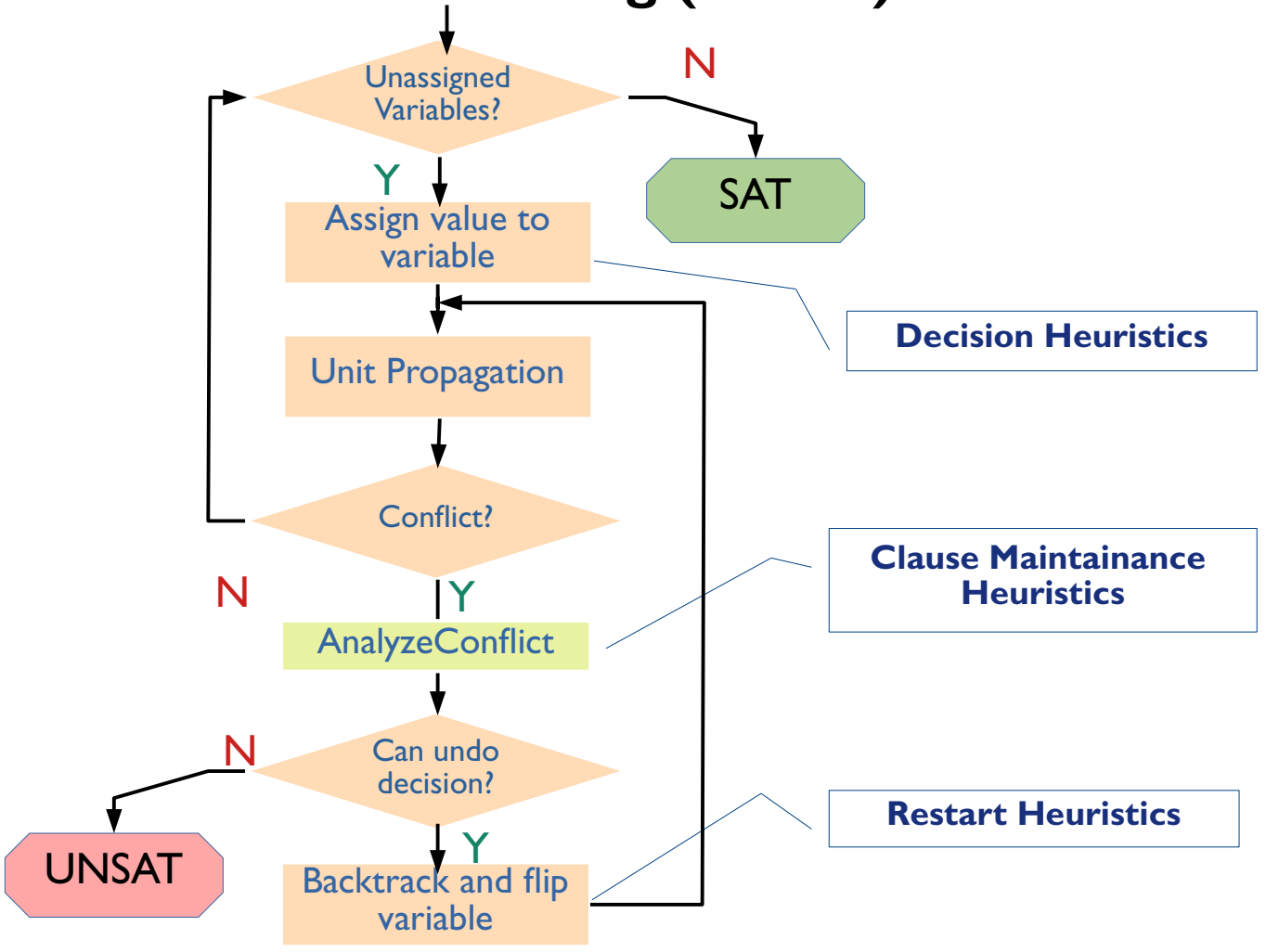
Conflict Driven Clause Learning (CDCL)



Conflict Driven Clause Learning (CDCL)



Conflict Driven Clause Learning (CDCL)



The curse of learnt clauses

- Learnt clauses are **very useful**
- But they consume memory and can **slowdown** other components of SAT solving
- Not practical to keep **all** the learnt clauses. We can keep around 5% of the learnt clauses.
- **Task** : predict whether the current clause will be useful in future.

The curse of learnt clauses

- Learnt clauses are **very useful**
- But they consume memory and can **slowdown** other components of SAT solving
- Not practical to keep **all** the learnt clauses. We can keep around 5% of the learnt clauses.
- **Task** : predict whether the current clause will be useful in future.
- **Popular heuristics** include
  - Delete **larger** clauses
  - Delete **less used** clauses
  - Delete clauses based on **Literal block distance**

Data-Driven Design of SAT solver

- View SAT solvers as composition of prediction engines
  - Branching
  - Clause learning
  - Memory management
  - Restarts

Data-Driven Design of SAT solver

- View SAT solvers as composition of prediction engines
  - Branching
  - Clause learning
  - Memory management
  - Restarts

CrystalBall Framework

CrystalBall Framework

- For inference, we want to do **supervised learning**
- For every clause, we need values of different features and **a label**
- The inference engine should **learn the model** to predict the label

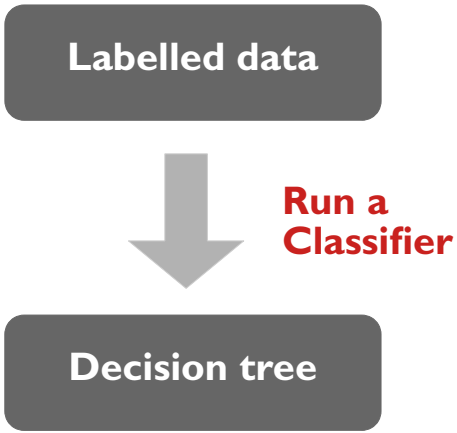
CrystalBall Framework

- For inference, we want to do **supervised learning**
- For every clause, we need values of different features and **a label**
- The inference engine should **learn the model** to predict the label

Labelled data

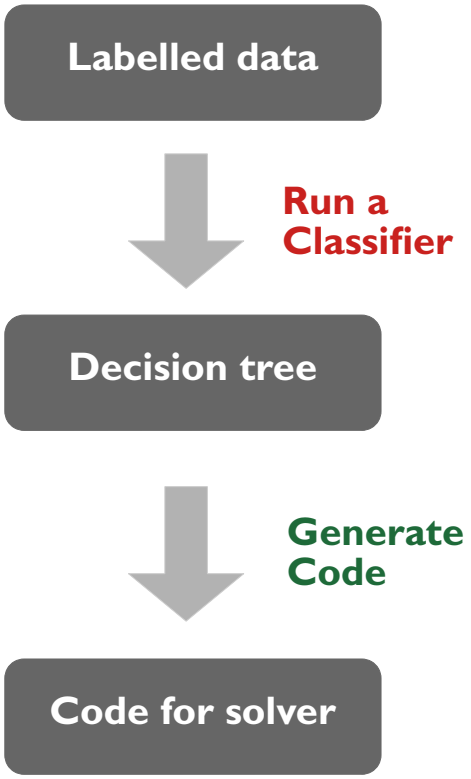
CrystalBall Framework

- For inference, we want to do **supervised learning**
- For every clause, we need values of different features and **a label**
- The inference engine should **learn the model** to predict the label



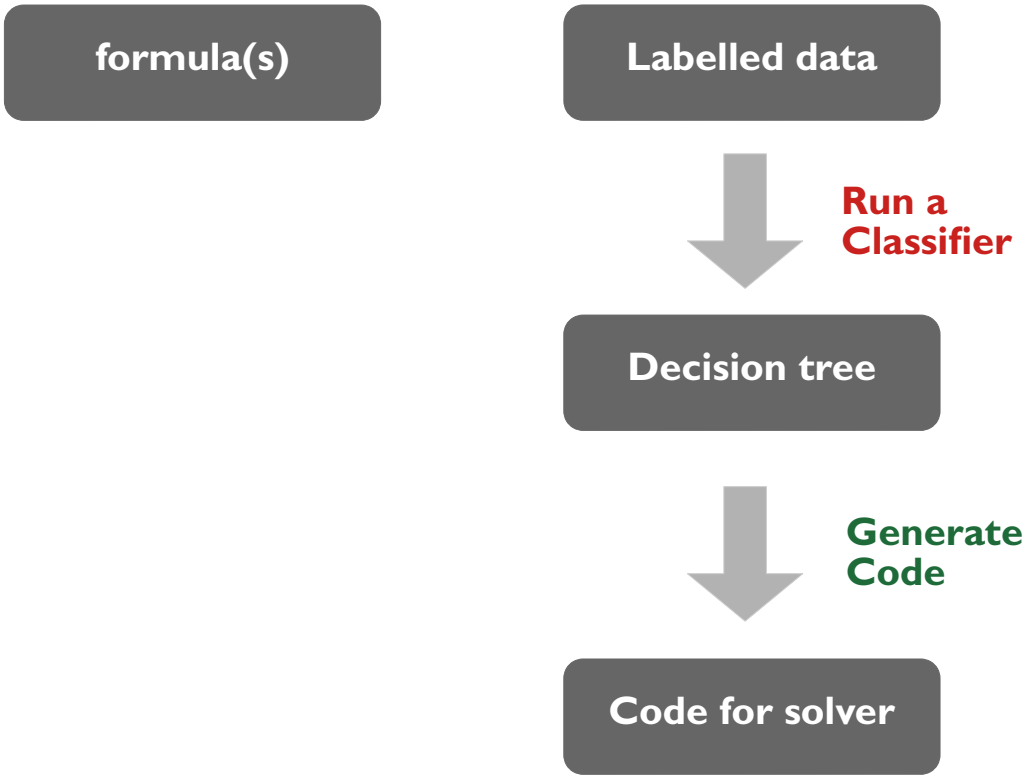
CrystalBall Framework

- For inference, we want to do **supervised learning**
- For every clause, we need values of different features and **a label**
- The inference engine should **learn the model** to predict the label



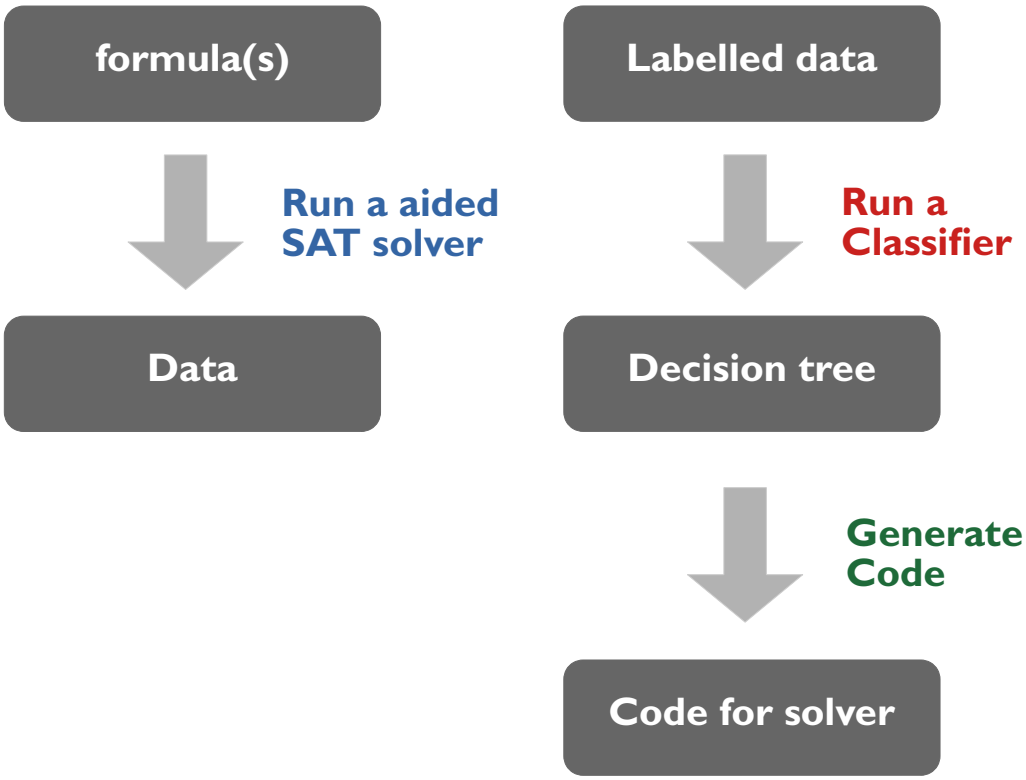
CrystalBall Framework

- For inference, we want to do **supervised learning**
- For every clause, we need values of different features and **a label**
- The inference engine should **learn the model** to predict the label



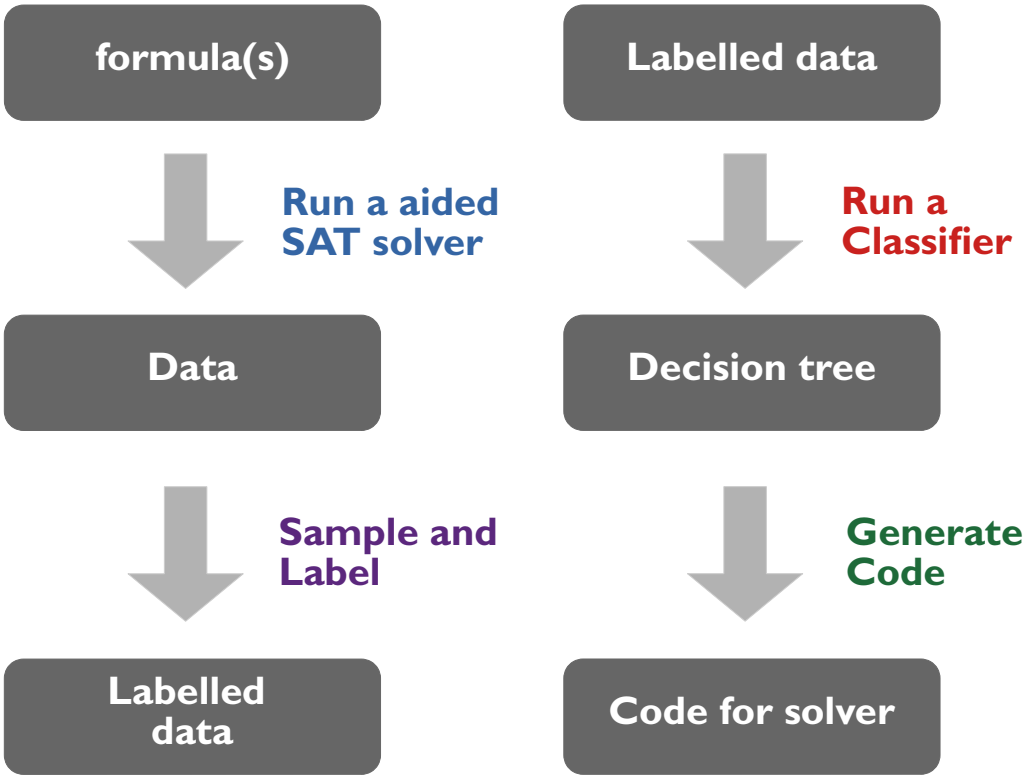
CrystalBall Framework

- For inference, we want to do **supervised learning**
- For every clause, we need values of different features and **a label**
- The inference engine should **learn the model** to predict the label



CrystalBall Framework

- For inference, we want to do **supervised learning**
- For every clause, we need values of different features and **a label**
- The inference engine should **learn the model** to predict the label



Feature Engineering / Data Collection

- **Modify our solver** CryptoMiniSAT to record different **features**, while the solver runs.
- Features include
  - properties of learnt clauses
  - state of solver
  - properties of formula

Feature Engineering / Data Collection

- **Modify our solver** CryptoMiniSAT to record different **features**, while the solver runs.
- Features include
  - properties of learnt clauses
  - state of solver
  - properties of formula

LBD	size	used_last_10k	activity

Feature Engineering / Data Collection

- **Modify our solver** CryptoMiniSAT to record different **features**, while the solver runs.
- Features include
  - properties of learnt clauses
  - state of solver
  - properties of formula
- Run on **UNSAT** instances

LBD	size	used_last_10k	activity

Feature Engineering / Data Collection

- **Modify our solver** CryptoMiniSAT to record different **features**, while the solver runs.
- Features include
  - properties of learnt clauses
  - state of solver
  - properties of formula
- Run on **UNSAT** instances

LBD	size	used_last_10k	activity
10	15	3	top half
7	10	1	bottom half
3	7	0	bottom half

Labeling

Task : predict whether the current clause will be useful in future.

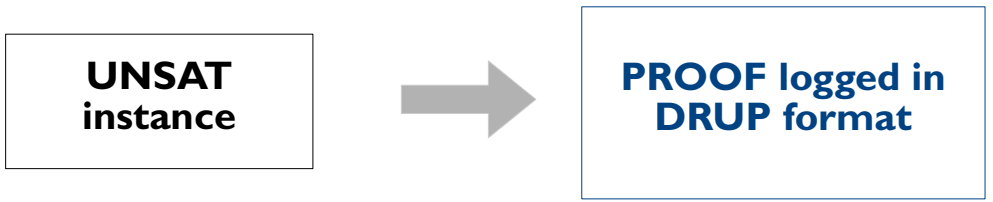
Labeling

Task : predict whether the current clause will be useful in future.

UNSAT instance

Labeling

Task : predict whether the current clause will be useful in future.



Labeling

Task : predict whether the current clause will be useful in future.

UNSAT instance

PROOF logged in DRUP format

```
p cnf
-2  3 0
 1  3 0
-1  2 0
-1 -2 0
 1 -2 0
 2 -3 0
 3  6 0
```

Labeling

Task : predict whether the current clause will be useful in future.

UNSAT instance

PROOF logged in DRUP format

```
p cnf
-2  3 0
 1  3 0
-1  2 0
-1 -2 0
 1 -2 0
 2 -3 0
 3  6 0
```

```
1 -2  3 0 0
2  1  3 0 0
3 -1  2 0 0
4 -1 -2 0 0
5  1 -2 0 0
6  2 -3 0 0
7 -2  0 4 5 0
8  3  0 1 2 3 0
9  0  6 7 8 0
```

Labeling

Task : predict whether the current clause will be useful in future.

UNSAT instance

PROOF logged in DRUP format

DRAT - trim

```
p cnf
-2  3 0
 1  3 0
-1  2 0
-1 -2 0
 1 -2 0
 2 -3 0
 3  6 0
```

```
1 -2  3 0 0
2  1  3 0 0
3 -1  2 0 0
4 -1 -2 0 0
5  1 -2 0 0
6  2 -3 0 0
7 -2  0 4 5 0
8  3  0 1 2 3 0
9  0  6 7 8 0
```

Backward pass to construct optimal proof.

Labeling

- Look at DRAT-trim's proof
- Check which learnt clauses are useful
- Label accordingly.

Labeling

- Look at DRAT-trim's proof
- Check which learnt clauses are useful
- Label accordingly.

glue	size	used_last_10k	activity
10	15	3	top half
7	10	1	bottom half
3	7	0	bottom half

Labeling

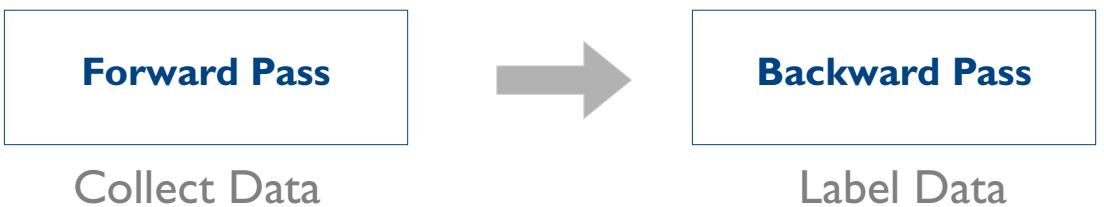
- Look at DRAT-trim's proof
- Check which learnt clauses are useful
- Label accordingly.

glue	size	used_last_10k	activity	label
10	15	3	top half	keep
7	10	1	bottom half	throw
3	7	0	bottom half	throw

Labeling

- Look at DRAT-trim's proof
- Check which learnt clauses are useful
- Label accordingly.

glue	size	used_last_10k	activity	label
10	15	3	top half	keep
7	10	1	bottom half	throw
3	7	0	bottom half	throw

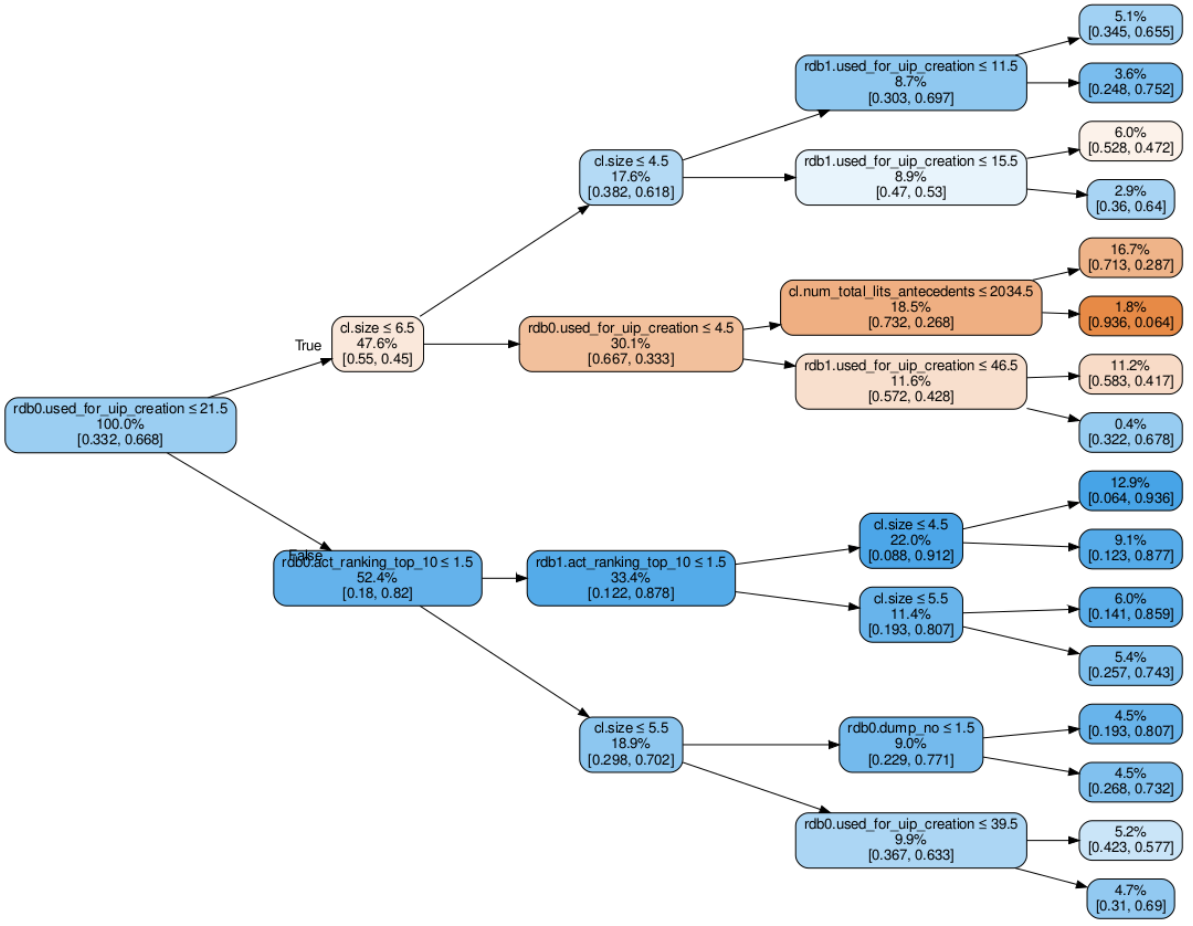


Inference Engine

glue	size	used_last_10k	label
10	15	3	keep
7	10	1	throw
3	7	0	throw

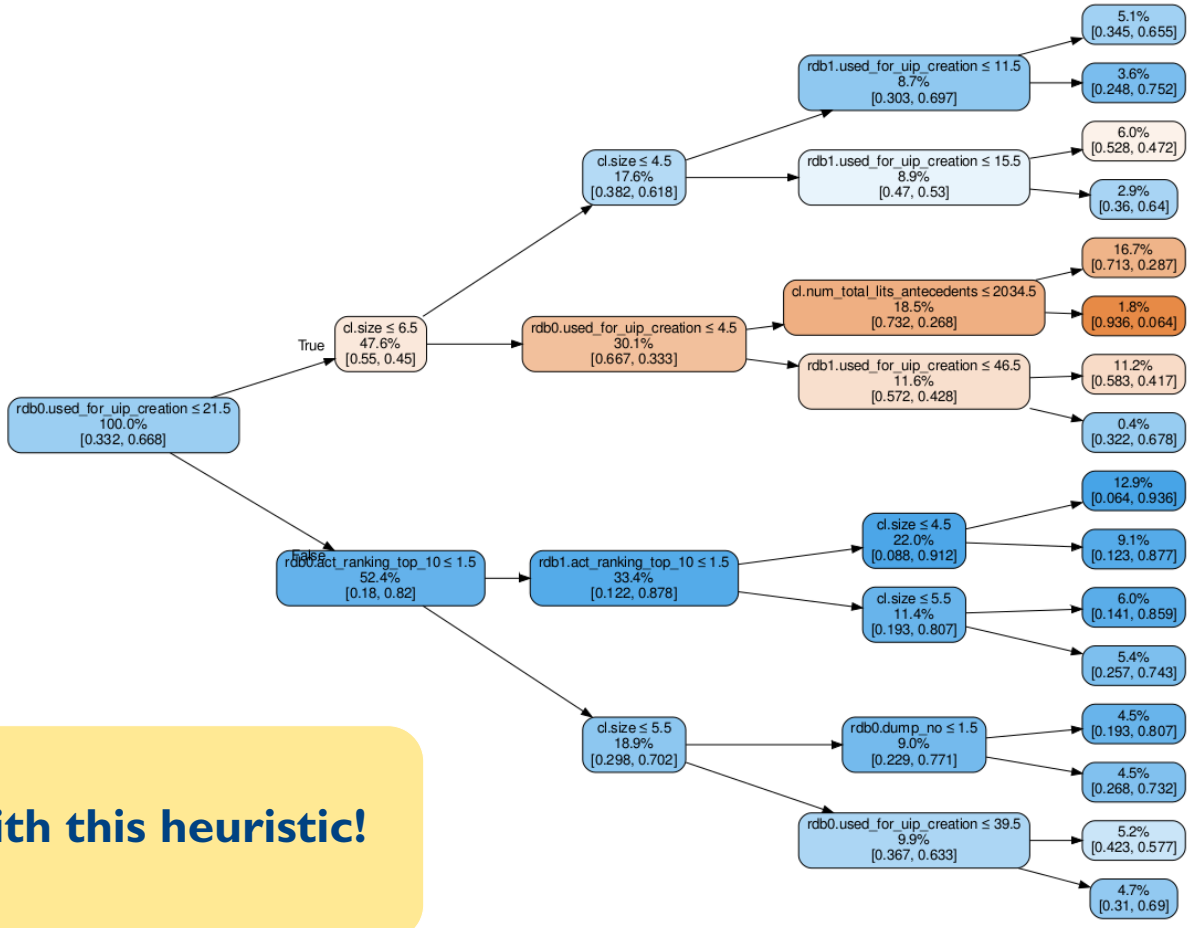
Inference Engine

glue	size	used_last_10k	label
10	15	3	keep
7	10	1	throw
3	7	0	throw



Inference Engine

glue	size	used_last_10k	label
10	15	3	keep
7	10	1	throw
3	7	0	throw



Create new solver with this heuristic!

Experimental Setup

- All the UNSAT instances from SAT 2014-17.
- The number of learnt clauses for different problems varied from few hundreds to millions
- In total, we had 422K data points.
- Tested on SAT competition 2019 benchmarks.

Experimental Setup

- All the UNSAT instances from SAT 2014-17.
- The number of learnt clauses for different problems varied from few hundreds to millions
- In total, we had 422K data points.
- Tested on SAT competition 2019 benchmarks.

on 400 SAT '19 instances

solver	# solved	Avg. Runtime*
CryptoMiniSat	291	9939
PredCryptoMiniSat	**299**	**9710**

Domain specific solver

comparing average runtime in seconds

	Benchmark	
Solver trained on	SATCompetition '19	SHA-1
SATCompetition	2440	1263
SHA-1	2805	1165

Domain specific solver

comparing average runtime in seconds

	Benchmark	
Solver trained on	SATCompetition '19	SHA-1
SATCompetition	2440	1263
SHA-1	2805	1165

	Benchmark
Solver trained on	AES
SATCompetition	1474
AES	1340

	Benchmark
Solver trained on	Grain
SATCompetition	1860
Grain	1973

Domain specific solver

comparing average runtime in seconds

	Benchmark	
Solver trained on	SATCompetition '19	SHA-1
SATCompetition	2440	1263
SHA-1	2805	1165

Rooms for improvement

- Sampling
- Labelling
- Different models to try

...

	Benchmark
Solver trained on	AES
SATCompetition	1474
AES	1340

	Benchmark
Solver trained on	Grain
SATCompetition	1860
Grain	1973

Conclusion

- First time, **white-box access** to SAT solvers.
- Democritizing SAT solver research : **easy to test new features**.
- Extend for **branching and restarts**.

Conclusion

- First time, **white-box access** to SAT solvers.
- Democritizing SAT solver research : **easy to test new features**.
- Extend for **branching and restarts**.
- Create **domain specific solvers.**

Conclusion

- First time, **white-box access** to SAT solvers.
- Democritizing SAT solver research : **easy to test new features**.
- Extend for **branching and restarts**.
- Create **domain specific solvers.**

Code : **meelgroup.github.io/crystalball**

\end{document}

An approach in solving Cryptographic Problems

An approach in solving Cryptographic Problems

Algebraic Cryptanalysis

An approach in solving Cryptographic Problems

Algebraic Cryptanalysis

Optimize SIMON Family

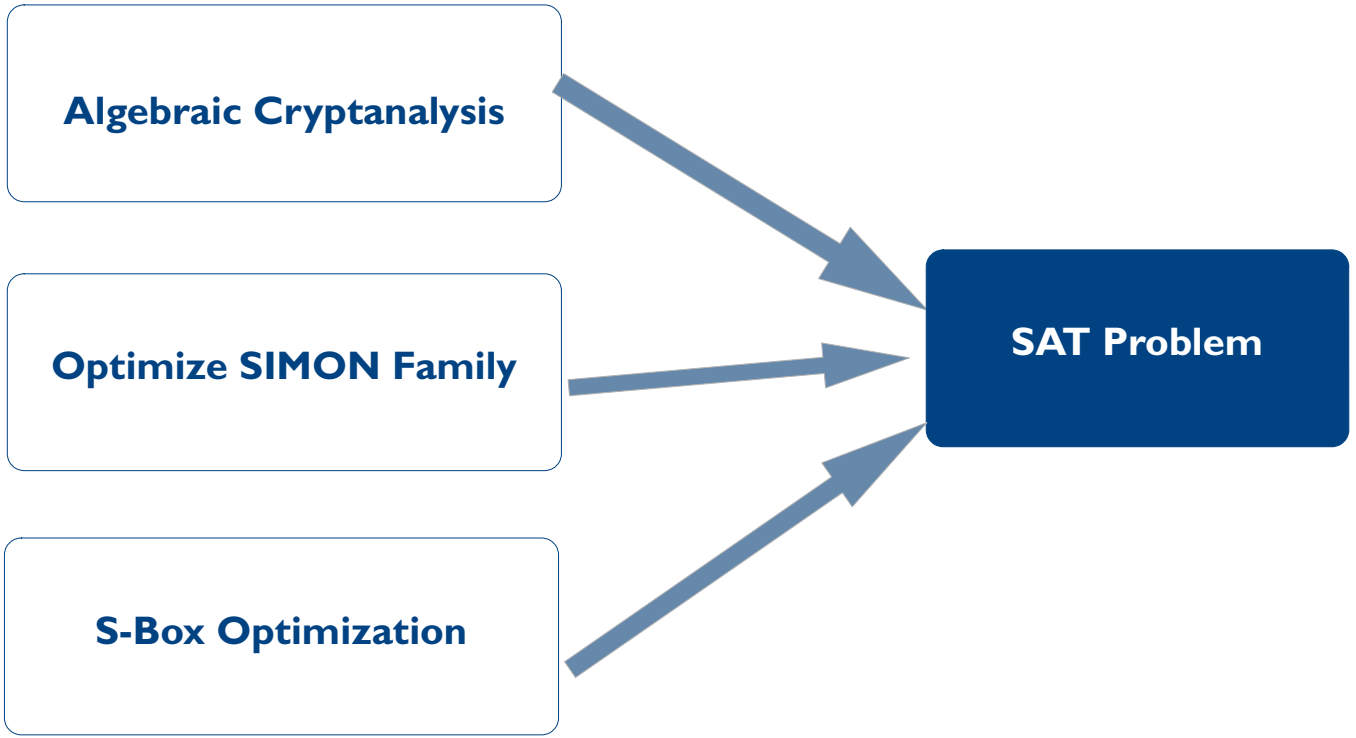
An approach in solving Cryptographic Problems

Algebraic Cryptanalysis

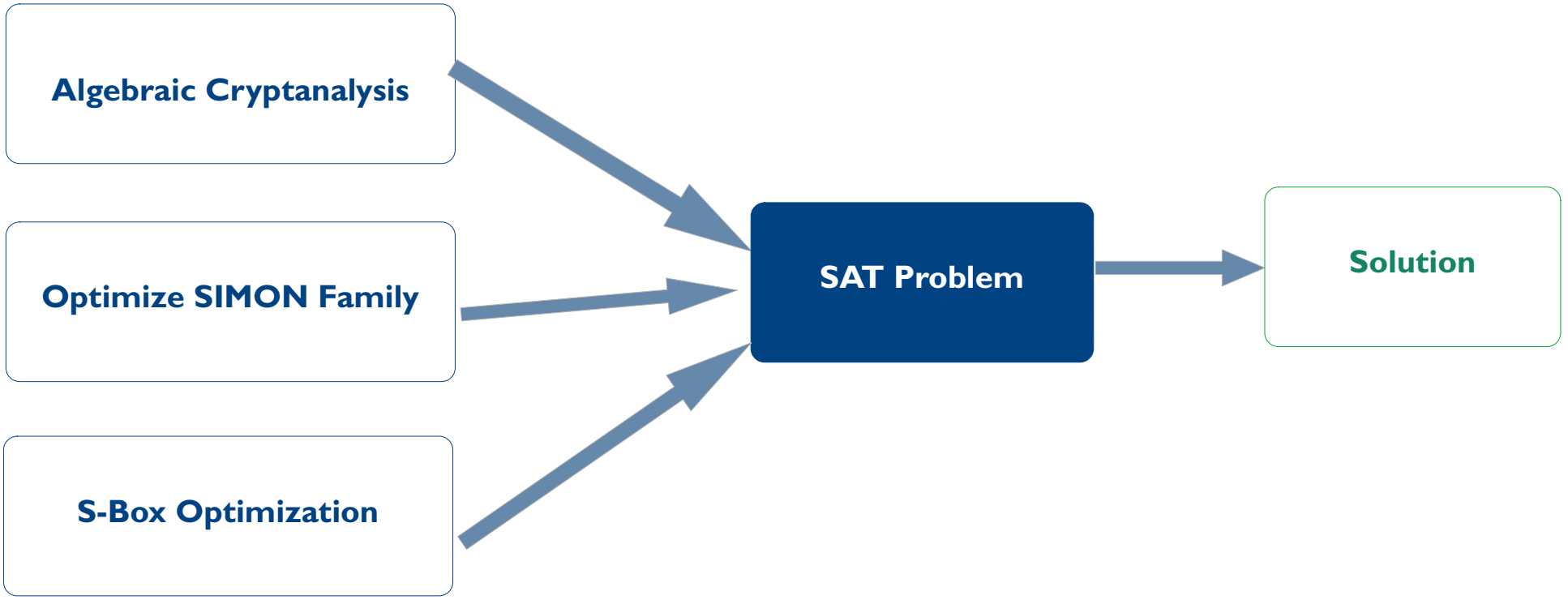
Optimize SIMON Family

S-Box Optimization

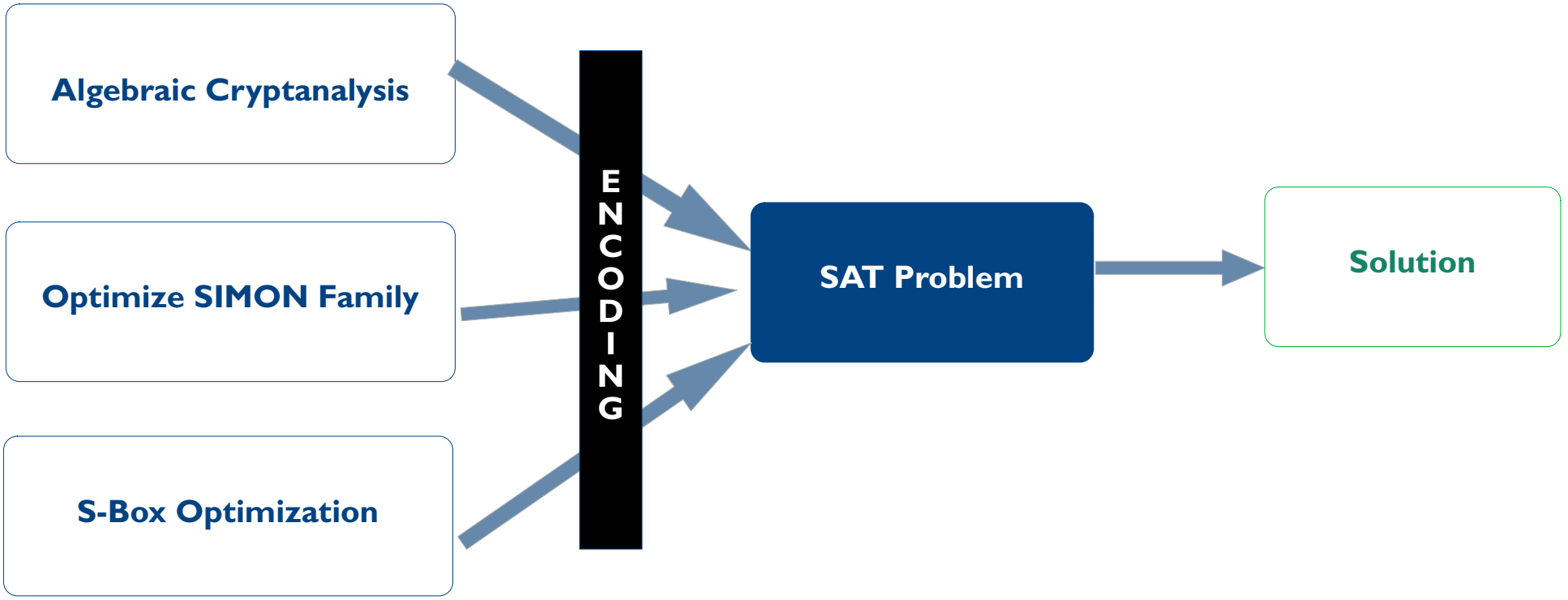
An approach in solving Cryptographic Problems



An approach in solving Cryptographic Problems



An approach in solving Cryptographic Problems



An approach in solving Cryptographic Problems