Progressive Generation of Canonical Sums of Products Using a SAT Solver

Ana Petkovska, Alan Mishchenko, David Novo, Muhsen Owaida, and Paolo Ienne

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Berkeley UNIVERSITY OF CALIFORNIA

June 10, 2016
Austin, Texas

Sum of Products (SOP) Representation

- **Sum of products (SOP)** is a two-level representation that can represent a Boolean function as a sum (OR, +) of cubes, $S = c_1 + \cdots + c_m$
- **Cube** c is a Boolean product (AND, $\cdot$) of literals, $c_i = l_1 \cdot \ldots \cdot l_k$
- **Literal** l is a variable v or its negation $\bar{v}$

Truth table

x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

minterms

On-set SOP

x_1	x_2	x_3	f
-	1	1	1
1	1	-	1

cubes

$f = x_2 x_3$

Sum of Products (SOP) Representation

- **Sum of products (SOP)** is a two-level representation that can represent a Boolean function as a sum (OR, +) of cubes, $S = c_1 + \cdots + c_m$
- **Cube** c is a Boolean product (AND, $\cdot$) of literals, $c_i = l_1 \cdot \ldots \cdot l_k$
- **Literal** l is a variable v or its negation $\bar{v}$

Truth table

x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

minterms

On-set SOP

x_1	x_2	x_3	f
-	1	1	1
1	1	-	1

cubes

$$f = x_2x_3 + x_1x_2$$

Sum of Products (SOP) Representation

- **Sum of products (SOP)** is a two-level representation that can represent a Boolean function as a sum (OR, +) of cubes, $S = c_1 + \cdots + c_m$
- **Cube** c is a Boolean product (AND, $\cdot$) of literals, $c_i = l_1 \cdot \ldots \cdot l_k$
- **Literal** l is a variable v or its negation $\bar{v}$

Truth table

x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

minterms

On-set SOP

x_1	x_2	x_3	f
-	1	1	1
1	1	-	1

$f = x_2x_3 + x_1x_2$

Off-set SOP

x_1	x_2	x_3	f
0	0	-	0
0	-	0	0
1	0	-	0

cubes

$\bar{f} = \bar{x}_1\,\bar{x}_2$

Sum of Products (SOP) Representation

- **Sum of products (SOP)** is a two-level representation that can represent a Boolean function as a sum (OR, +) of cubes, $S = c_1 + \cdots + c_m$
- **Cube** c is a Boolean product (AND, $\cdot$) of literals, $c_i = l_1 \cdot \ldots \cdot l_k$
- **Literal** l is a variable v or its negation $\bar{v}$

Truth table

x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

minterms

On-set SOP

x_1	x_2	x_3	f
-	1	1	1
1	1	-	1

$f = x_2x_3 + x_1x_2$

Off-set SOP

x_1	x_2	x_3	f
0	0	-	0
0	-	0	0
1	0	-	0

cubes

$\bar{f} = \bar{x}_1\,\bar{x}_2 + \bar{x}_1\,\bar{x}_3$

Sum of Products (SOP) Representation

- **Sum of products (SOP)** is a two-level representation that can represent a Boolean function as a sum (OR, +) of cubes, $S = c_1 + \cdots + c_m$
- **Cube** c is a Boolean product (AND, $\cdot$) of literals, $c_i = l_1 \cdot \ldots \cdot l_k$
- **Literal** l is a variable v or its negation $\bar{v}$

Truth table

x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

minterms

On-set SOP

x_1	x_2	x_3	f
-	1	1	1
1	1	-	1

$f = x_2x_3 + x_1x_2$

Off-set SOP

x_1	x_2	x_3	f
0	0	-	0
0	-	0	0
1	0	-	0

cubes

$\bar{f} = \bar{x}_1\,\bar{x}_2 + \bar{x}_1\,\bar{x}_3 + x_1\bar{x}_2$

Sum of Products (SOP) Representation

- **Sum of products (SOP)** is a two-level representation that can represent a Boolean function as a sum (OR, +) of cubes, $S = c_1 + \cdots + c_m$
- **Cube** c is a Boolean product (AND, $\cdot$) of literals, $c_i = l_1 \cdot \ldots \cdot l_k$
- **Literal** l is a variable v or its negation $\bar{v}$

Truth table

x_1	x_2	x_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

minterms

On-set SOP

x_1	x_2	x_3	f
-	1	1	1
1	1	-	1

$f = x_2x_3 + x_1x_2$

SOP size: 2 cubes, 4 literals

smaller SOP

Off-set SOP

x_1	x_2	x_3	f
0	0	-	0
0	-	0	0
1	0	-	0

cubes

$\bar{f} = \bar{x}_1\,\bar{x}_2 + \bar{x}_1\,\bar{x}_3 + x_1\bar{x}_2$

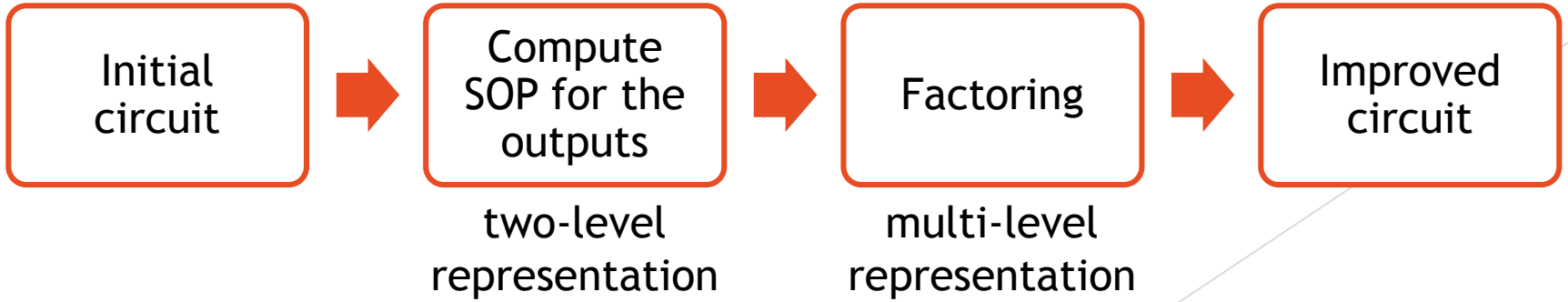
SOP size: 3 cubes, 6 literals

Use of SOPs

- Mapping into Programmable Logic Arrays (PLAs)

Supported in many tools for logic optimization (in academia and industry)

- Delay optimization in technology independent synthesis and technology mapping
- Test generation
- Multi-level logic synthesis for global circuit restructuring (case-study)



Existing SOP Generation and Minimization

- From Binary Decision Diagrams (BDDs)
  - **Problem 1**: BDD generation is not scalable for some functions
    - The BDD size is exponential in the number of input variables → BDD memory explosion problem
  - **Problem 2**: Incompatible with incremental applications
    - The complete BDD should be build before converting it to SOP
- ESPRESSO style minimizers: version enhanced by SAT solving
  - **Problem 1**: Requires as input a computed SOP: do not generate a new SOP from a multi-level representation
  - **Problem 2**: The existing SOP generation is based on enumeration on satisfying assignments

Existing SOP Generation and Minimization

- From Binary Decision Diagrams (BDDs)
  - **Problem 1**: BDD generation is not scalable for some functions
    - The BDD size is exponential in the number of input variables → BDD memory explosion problem
  - **Problem 2**: Incompatible with incremental applications
    - The complete BDD should be build before converting it to SOP
- ESPRESSO style minimizers: version enhanced by SAT solving
  - **Problem 1**: Requires as input a computed SOP: do not generate a new SOP from a multi-level representation
  - **Problem 2**: The existing SOP generation is based on enumeration on satisfying assignments

Solution?

SAT-based SOP Generation and Minimization

- From Binary Decision Diagrams (BDDs)

Feasibility of computation

BDD generation is not scalable for some functions

size is exponential in the number of input variables → BDD memory explosion problem

Progressive generation

Incompatible with incremental applications

Partial SOPs

plete BDD should be build before converting it to SOP

- ESPRESSO style minimizers: version enhanced by SAT solving

Completely based on SAT solving

Requires as input a computed SOP: do not generate a new SOP from a multi-level
tion

The existing SOP generation is based on enumeration on satisfying assignments

Solution?

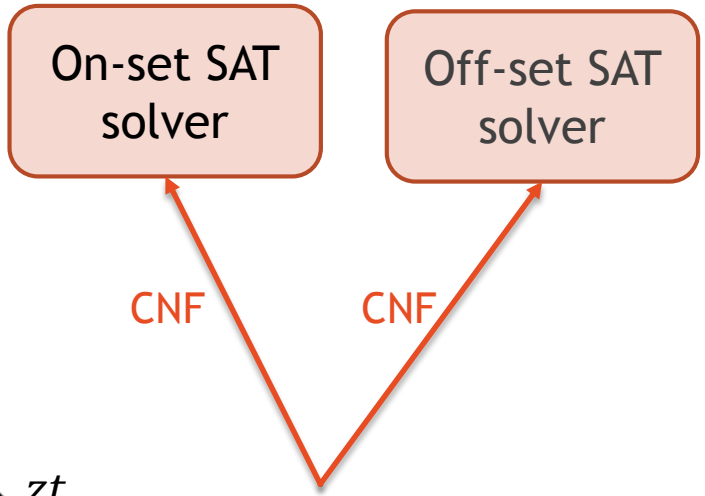
SAT-based SOP Generation and Minimization

- Completely based on SAT solvers
- **Progressive generation**: cube by cube (generated and minimized)
  - Generation of a **partial SOP**
  - Prediction of **feasibility of computation**
- SAT-based generation of **canonical SOPs**
  - Unique SOP for *a given function* and *variable order*, independent of
    - The initial structure of the circuit
    - CNF generation algorithm
    - The used SAT solver
    - Operating system
- The generated SOPs are **irredundant**
  - No literal and no cube can be removed without changing the function

Outline

- SAT-based SOP Generation
- Methods for Runtime Improvement
- Experimental Results
- Conclusion

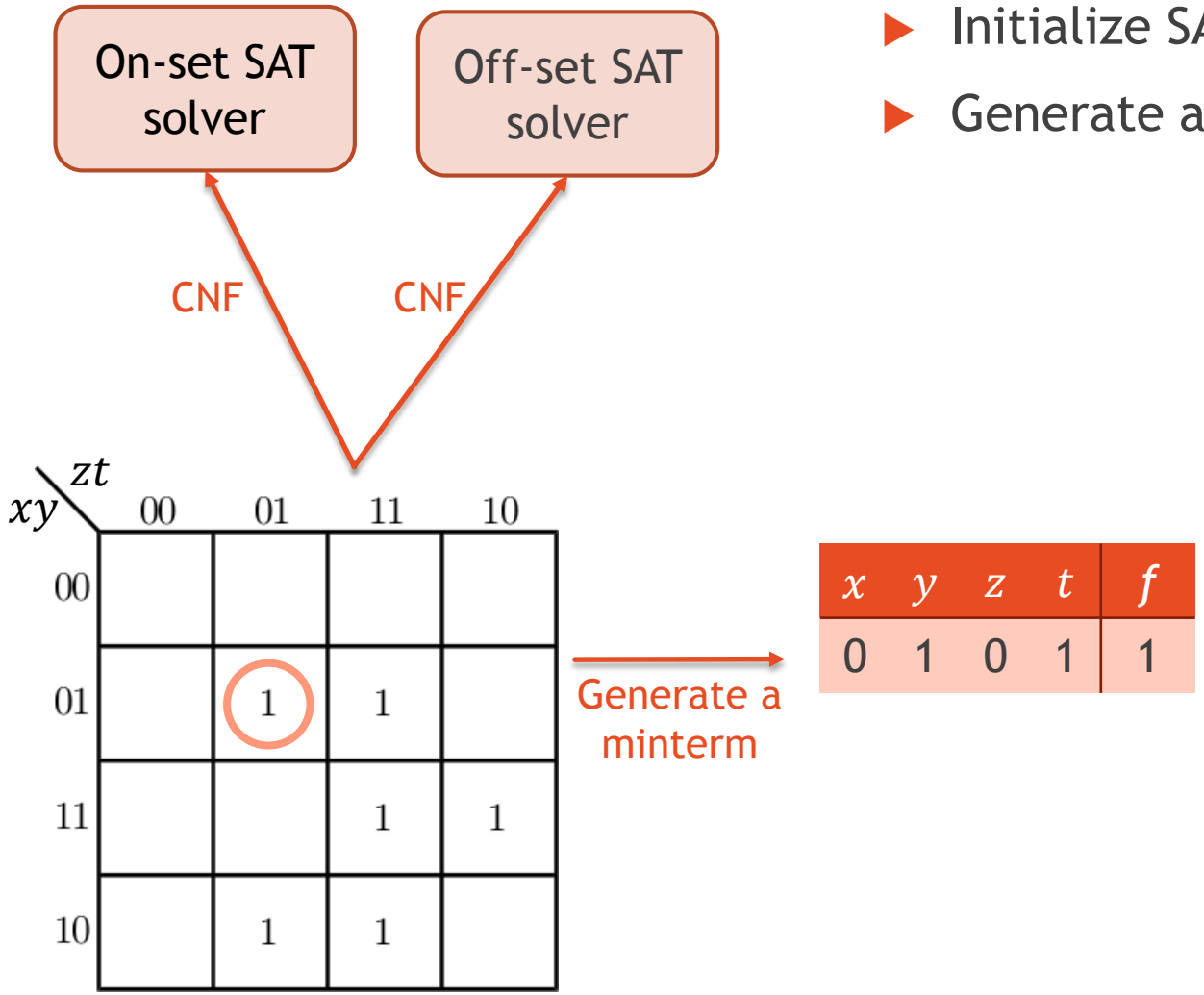
Progressive SAT-based SOP Generation: Global view



xy \ zt	00	01	11	10
00				
01		1	1	
11			1	1
10		1	1	

- Initialize SAT solvers

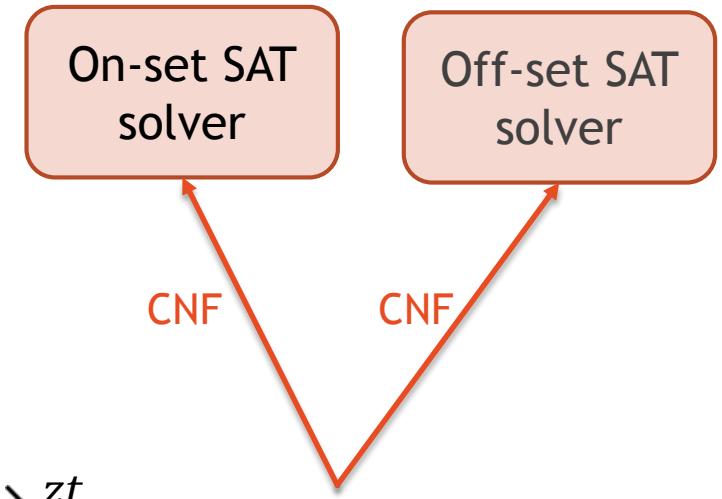
Progressive SAT-based SOP Generation: Global view



x	y	z	t	f
0	1	0	1	1

- Initialize SAT solvers
- Generate a minterm

Progressive SAT-based SOP Generation: Global view



- Initialize SAT solvers
- Generate a minterm
- Expand the minterm to a cube

xy \ zt	00	01	11	10
00				
01		1	1	
11			1	1
10		1	1	

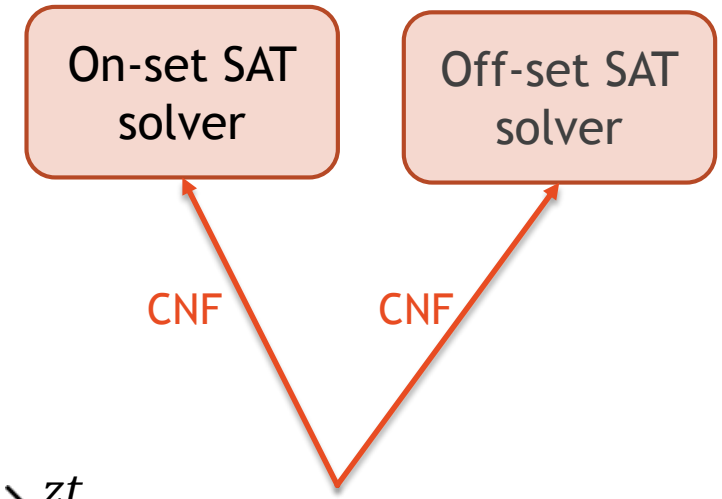
Generate a minterm

x	y	z	t	f
0	1	0	1	1

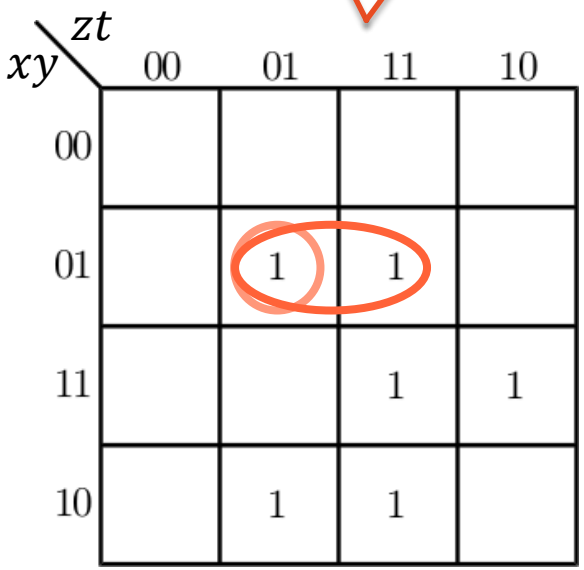
Expand to a cube

x	y	z	t	f
0	1	-	1	1

Progressive SAT-based SOP Generation: Global view



- Initialize SAT solvers
- Generate a minterm
- Expand the minterm to a cube
- Add the cube as a blocking clause



Generate a minterm

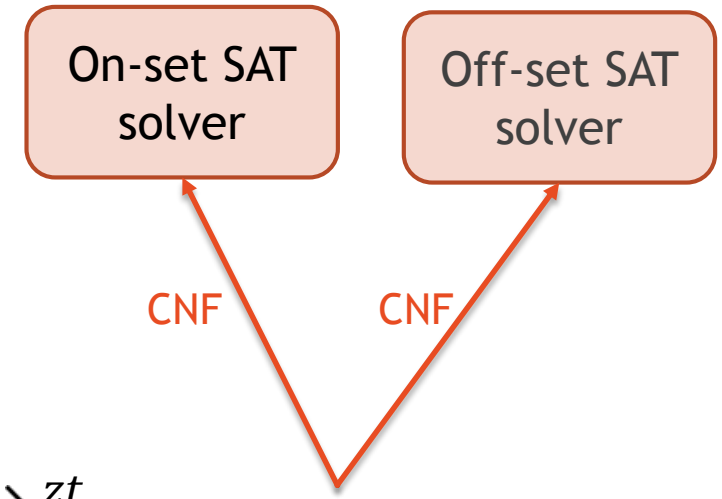
x	y	z	t	f
0	1	0	1	1

Expand to a cube

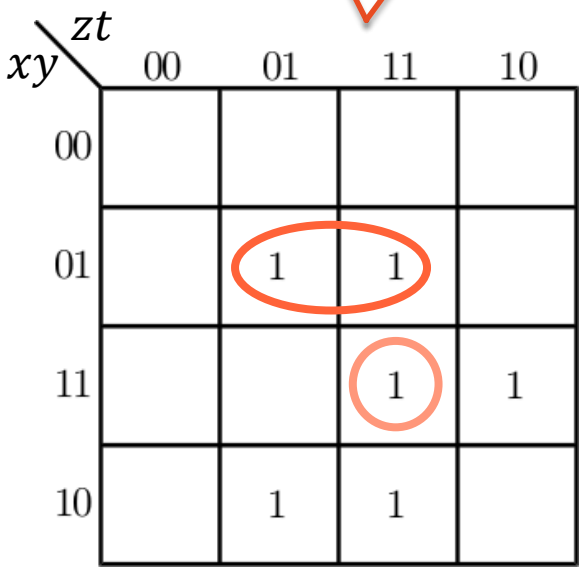
x	y	z	t	f
0	1	-	1	1

$\bar{x}yt = (x + \bar{y} + \bar{t})$

Progressive SAT-based SOP Generation: Global view



- Initialize SAT solvers
- Generate a minterm
- Expand the minterm to a cube
- Add the cube as a blocking clause



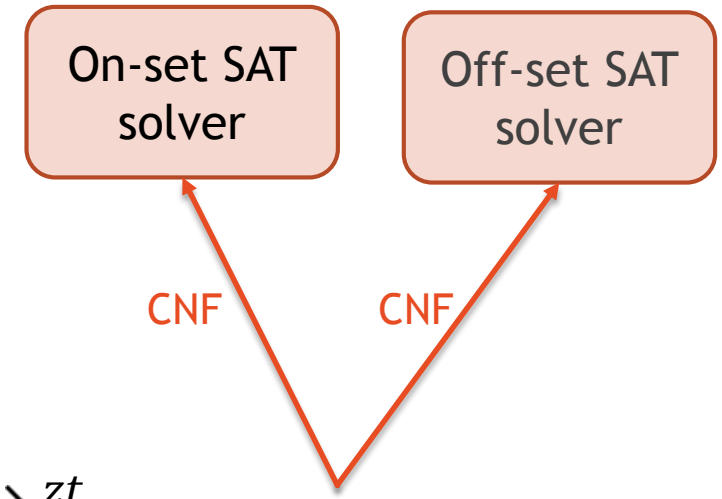
Generate a minterm

x	y	z	t	f
0	1	0	1	1
1	1	1	1	1

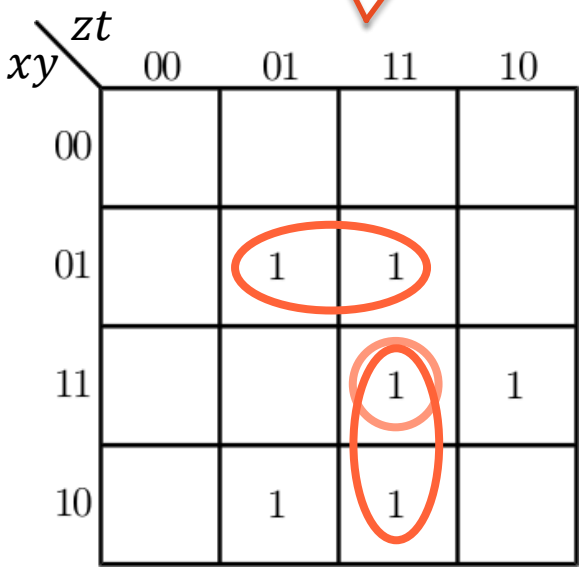
Expand to a cube

x	y	z	t	f
0	1	-	1	1

Progressive SAT-based SOP Generation: Global view



- Initialize SAT solvers
- Generate a minterm
- Expand the minterm to a cube
- Add the cube as a blocking clause



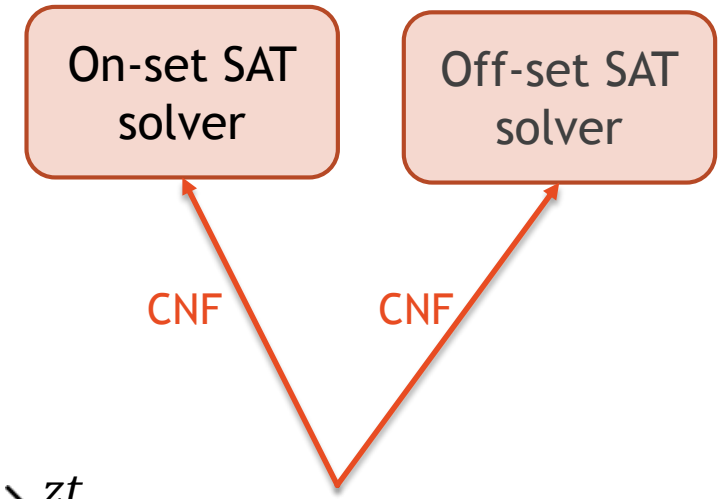
Generate a minterm

x	y	z	t	f
0	1	0	1	1
1	1	1	1	1

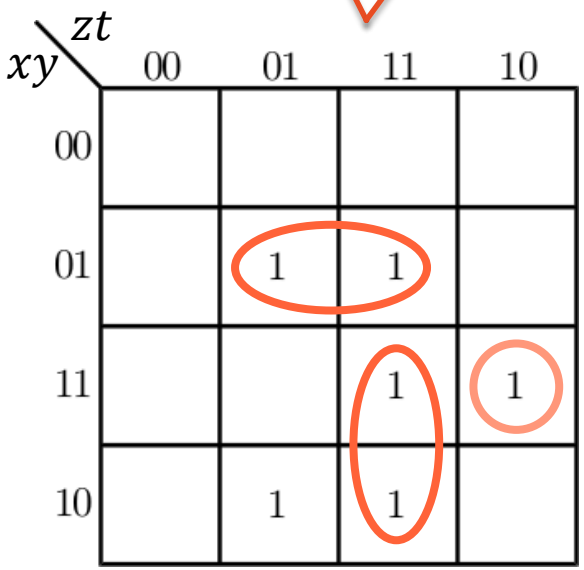
Expand to a cube

x	y	z	t	f
0	1	-	1	1
1	-	1	1	1

Progressive SAT-based SOP Generation: Global view



- Initialize SAT solvers
- Generate a minterm
- Expand the minterm to a cube
- Add the cube as a blocking clause



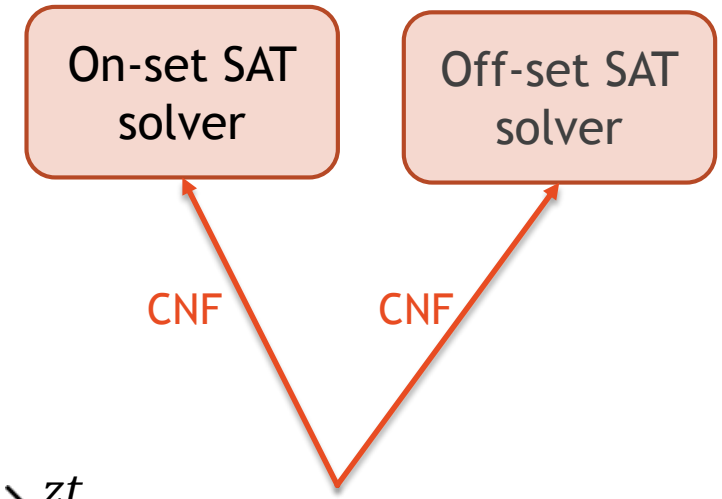
Generate a minterm

x	y	z	t	f
0	1	0	1	1
1	1	1	1	1
1	1	1	0	1

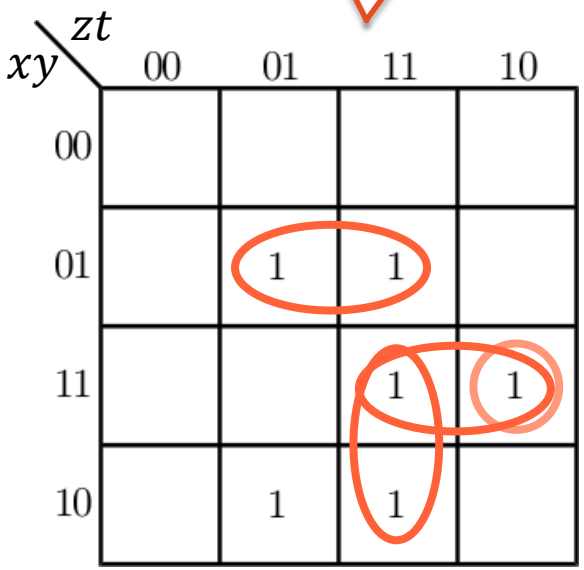
Expand to a cube

x	y	z	t	f
0	1	-	1	1
1	-	1	1	1

Progressive SAT-based SOP Generation: Global view



- Initialize SAT solvers
- Generate a minterm
- Expand the minterm to a cube
- Add the cube as a blocking clause



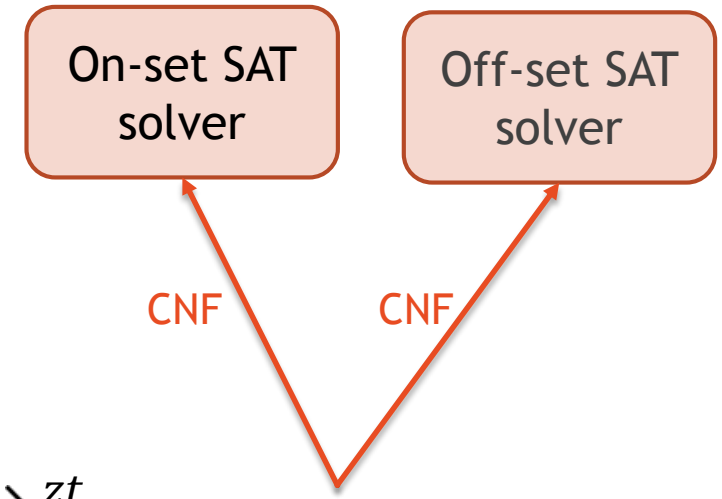
Generate a minterm

x	y	z	t	f
0	1	0	1	1
1	1	1	1	1
1	1	1	0	1

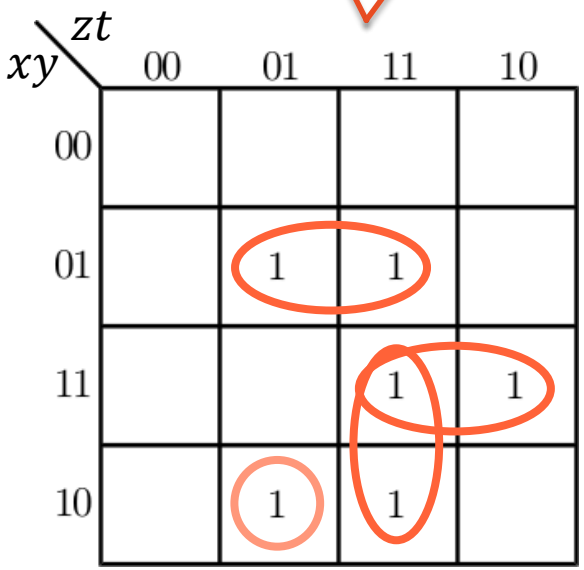
Expand to a cube

x	y	z	t	f
0	1	-	1	1
1	-	1	1	1
1	1	1	-	1

Progressive SAT-based SOP Generation: Global view



- Initialize SAT solvers
- Generate a minterm
- Expand the minterm to a cube
- Add the cube as a blocking clause



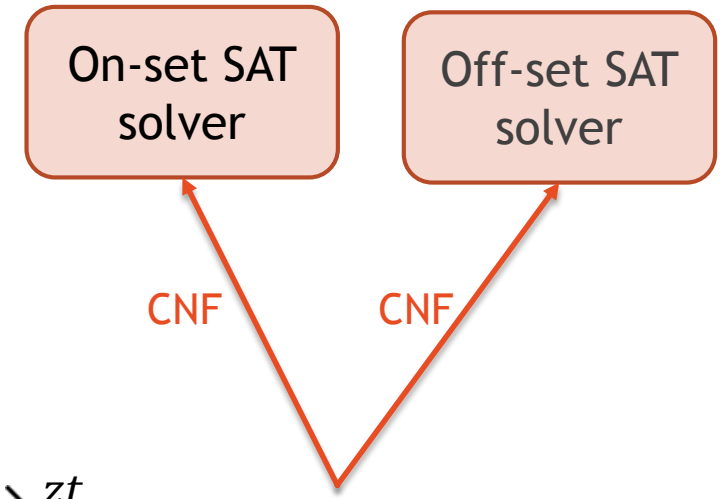
Generate a minterm

x	y	z	t	f
0	1	0	1	1
1	1	1	1	1
1	1	1	0	1
1	0	0	1	1

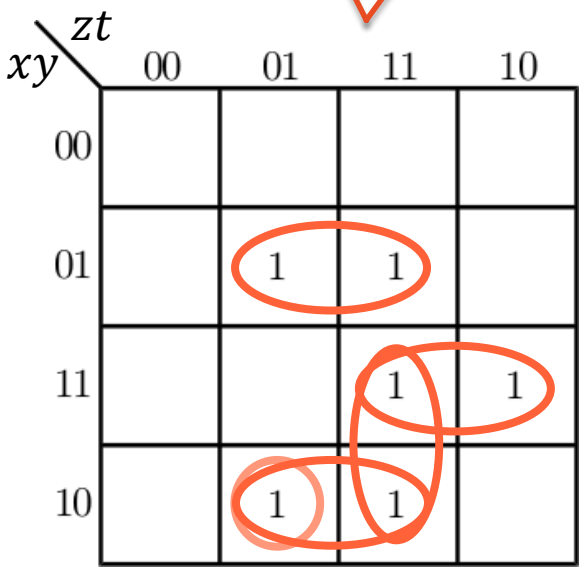
Expand to a cube

x	y	z	t	f
0	1	-	1	1
1	-	1	1	1
1	1	1	-	1

Progressive SAT-based SOP Generation: Global view



- Initialize SAT solvers
- Generate a minterm
- Expand the minterm to a cube
- Add the cube as a blocking clause



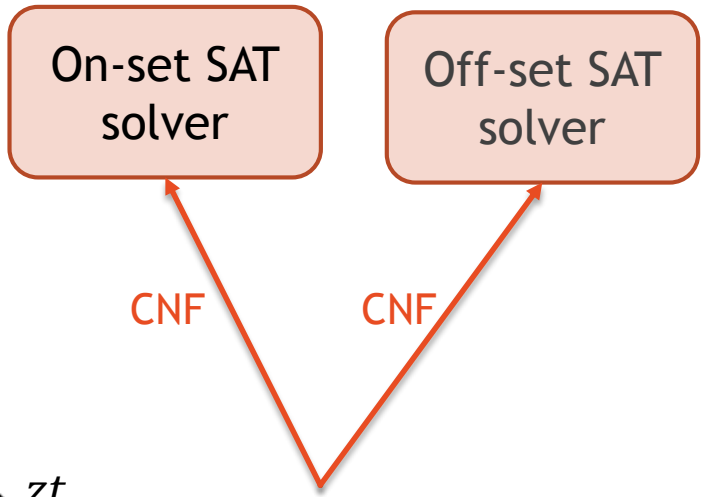
Generate a minterm

x	y	z	t	f
0	1	0	1	1
1	1	1	1	1
1	1	1	0	1
1	0	0	1	1

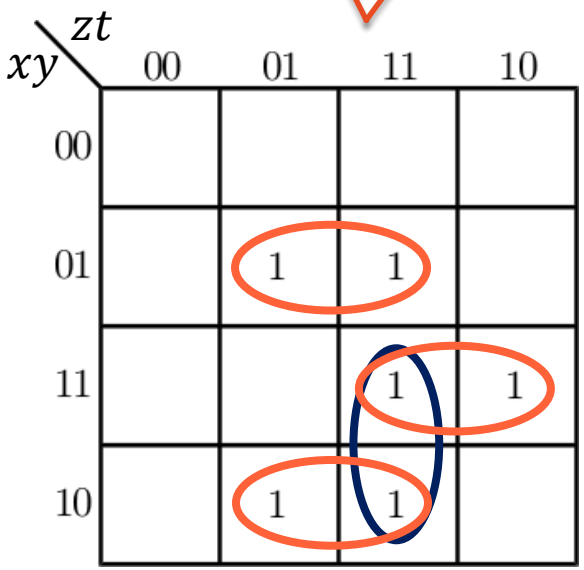
Expand to a cube

x	y	z	t	f
0	1	-	1	1
1	-	1	1	1
1	1	1	-	1
1	0	-	1	1

Progressive SAT-based SOP Generation: Global view



- Initialize SAT solvers
- Generate a minterm
- Expand the minterm to a cube
- Add the cube as a blocking clause
- Remove redundant cubes



Generate a minterm

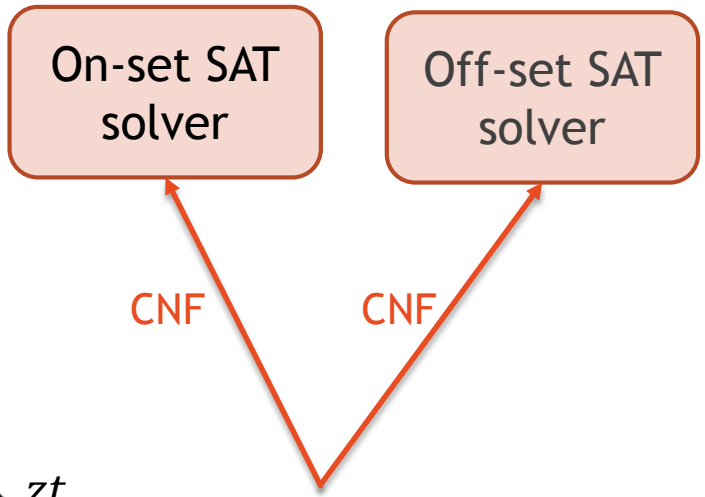
x	y	z	t	f
0	1	0	1	1
1	1	1	1	1
1	1	1	0	1
1	0	0	1	1

Expand to a cube

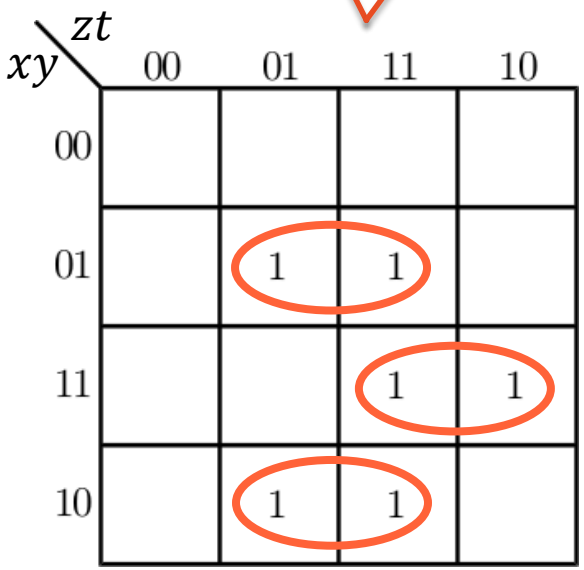
(Redundant?) On-set SOP

x	y	z	t	f
0	1	-	1	1
1	-	1	1	1
1	1	1	0	1
1	0	-	1	1

Progressive SAT-based SOP Generation: Global view



- Initialize SAT solvers
- Generate a minterm
- Expand the minterm to a cube
- Add the cube as a blocking clause
- Remove redundant cubes



Generate a minterm

x	y	z	t	f
0	1	0	1	1
1	1	1	1	1
1	1	1	0	1
1	0	0	1	1

Expand to a cube

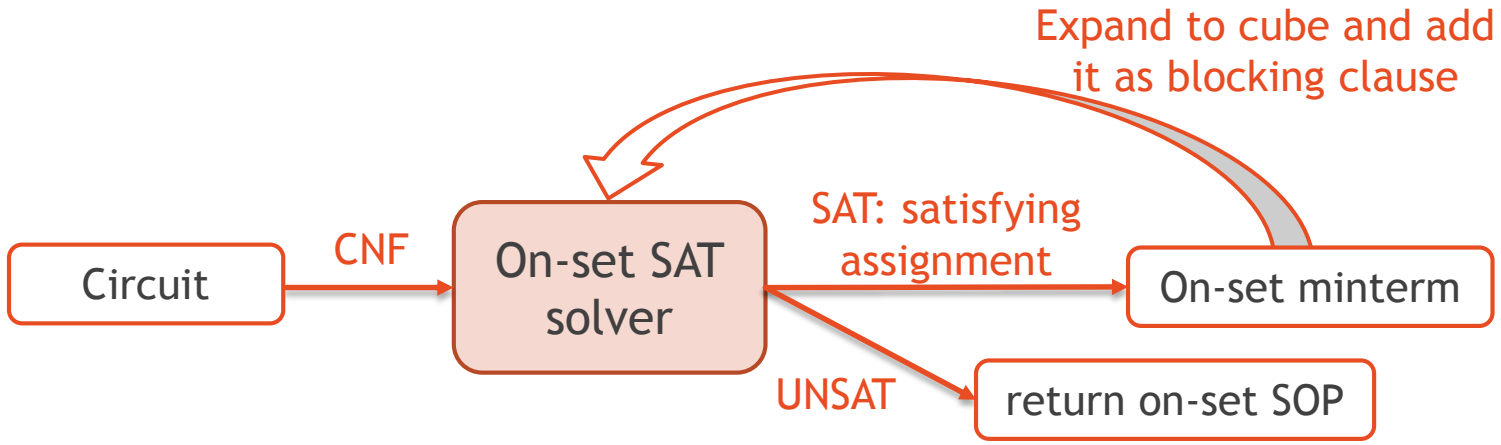
Irredundant On-set SOP

x	y	z	t	f
0	1	-	1	1
1	-	1	1	1
1	0	-	1	1

Outline

- SAT-based SOP Generation
  - Generation of Minterms
  - Expansion of Minterms to Cubes
  - Removing Redundant Cubes
- Methods for Runtime Improvement
- Experimental Results
- Conclusion

Generation of Minterms

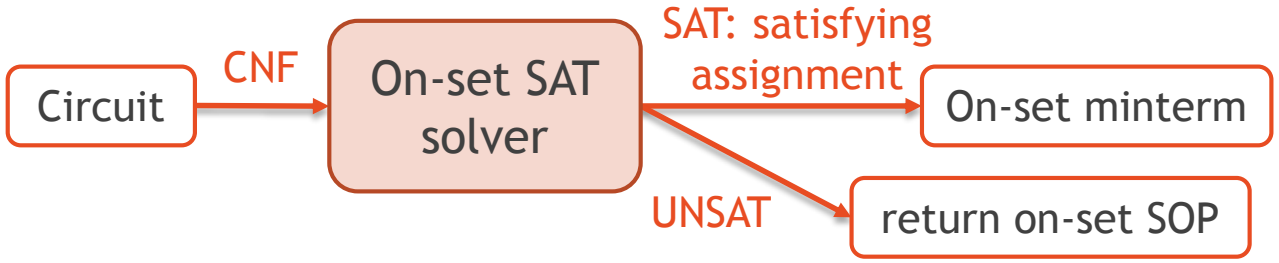


- *Non-canonical SOPs*: A SAT assignment
- *Canonical SOPs*: The lexicographically smallest SAT assignment under the variable order
  - Use the LEXSAT algorithm
  - Deterministic algorithm: generate the same minterms in the same order

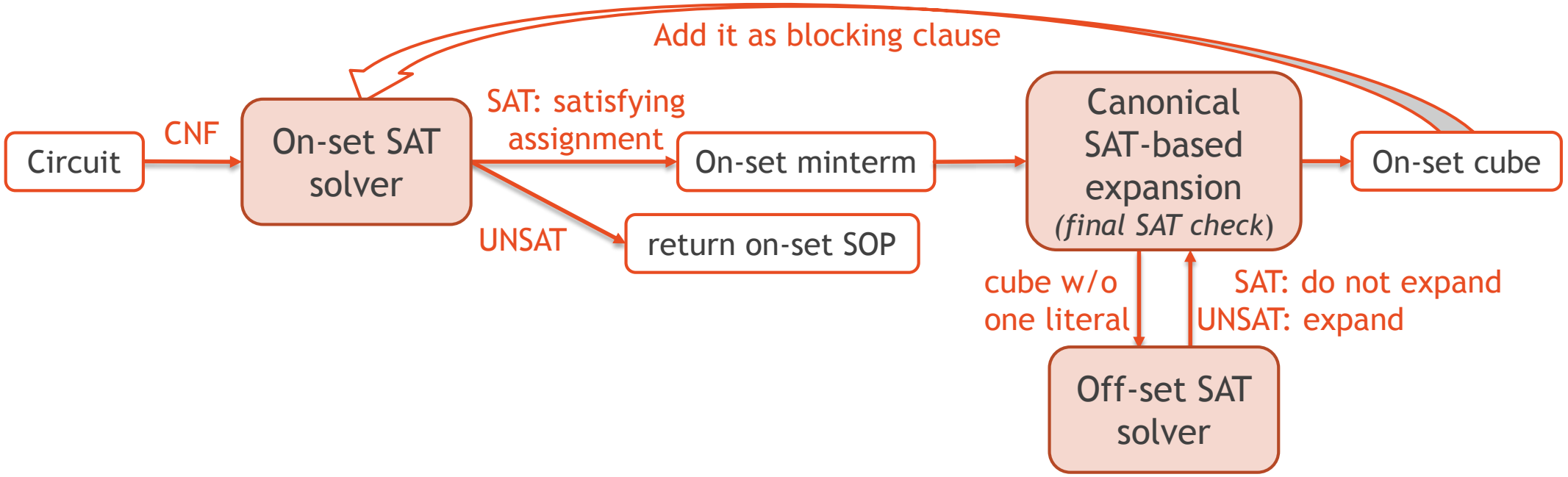
Outline

- SAT-based SOP Generation
  - Generation of Minterms
  - Expansion of Minterms to Cubes
  - Removing Redundant Cubes
- Methods for Runtime Improvement
- Experimental Results
- Conclusion

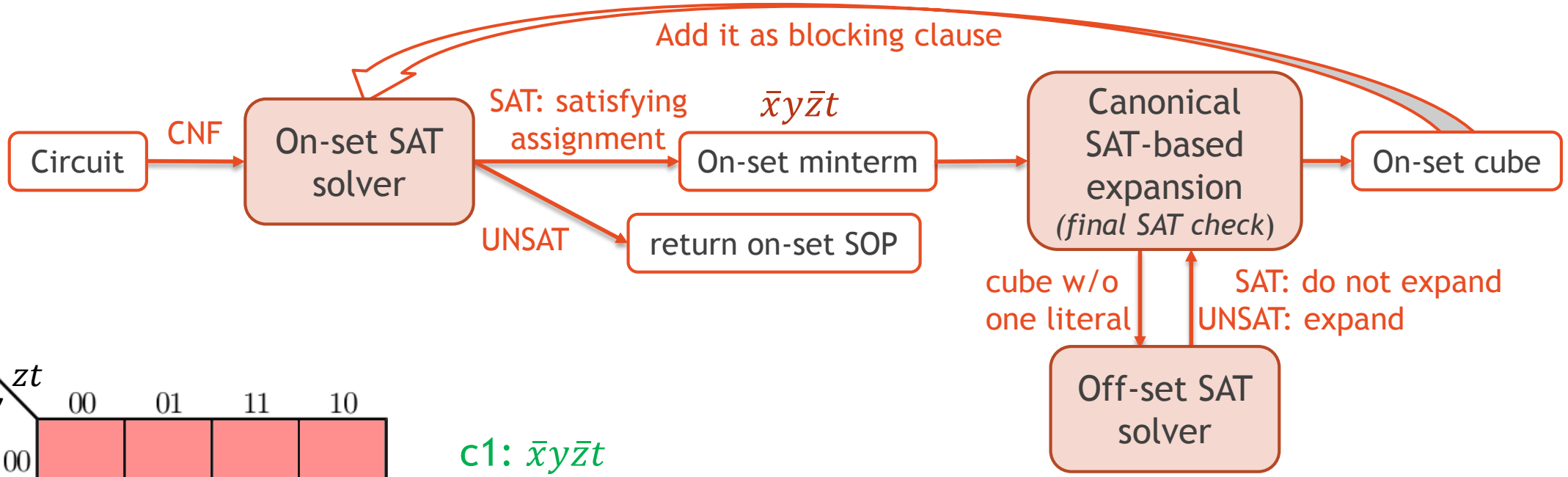
Expansion of Minterms to Cubes

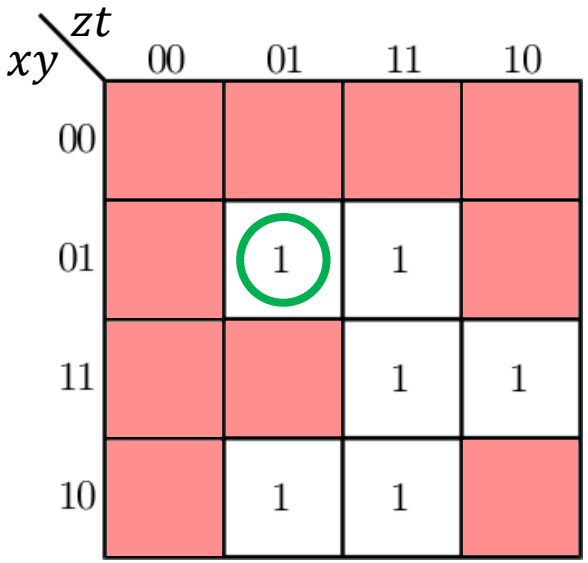


Expansion of Minterms to Cubes



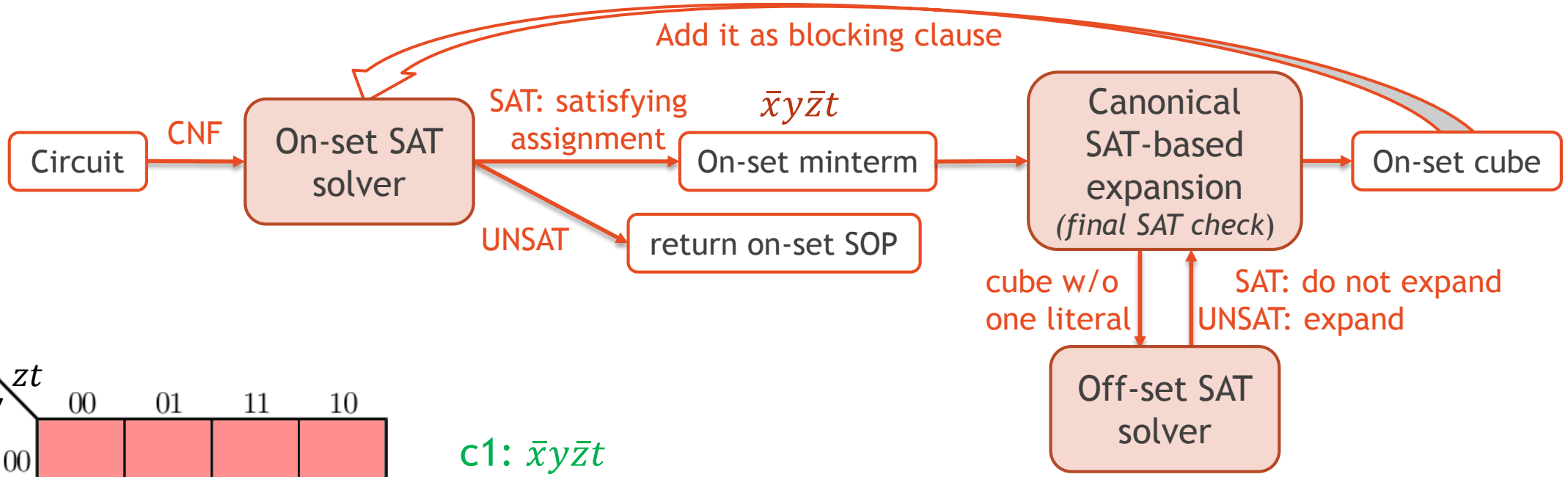
Expansion of Minterms to Cubes

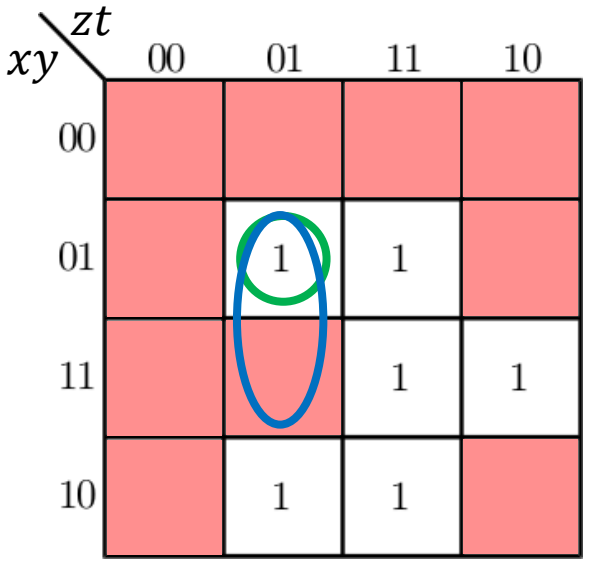




c1: $\bar{x}y\bar{z}t$

Expansion of Minterms to Cubes

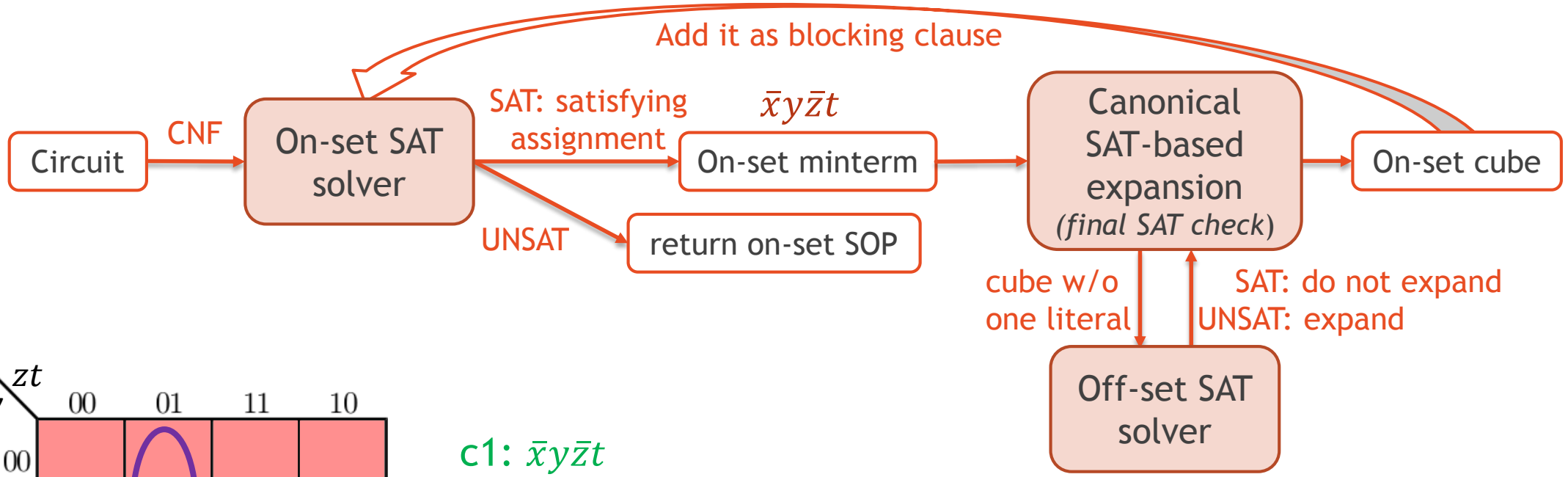


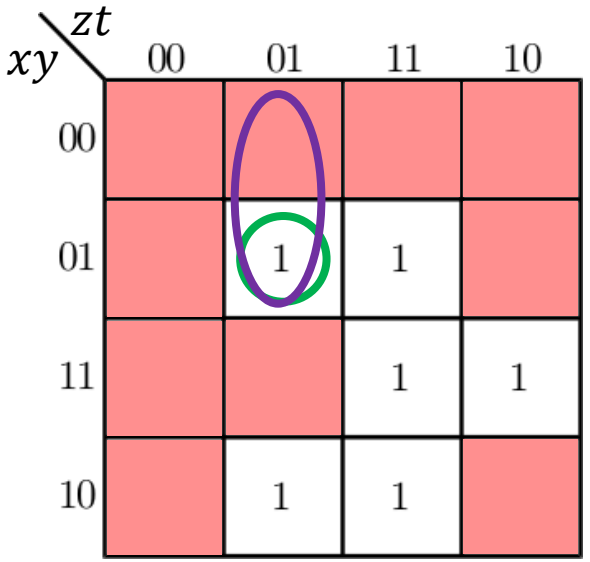


c1: $\bar{x}y\bar{z}t$

c2: $-y\bar{z}t$

Expansion of Minterms to Cubes



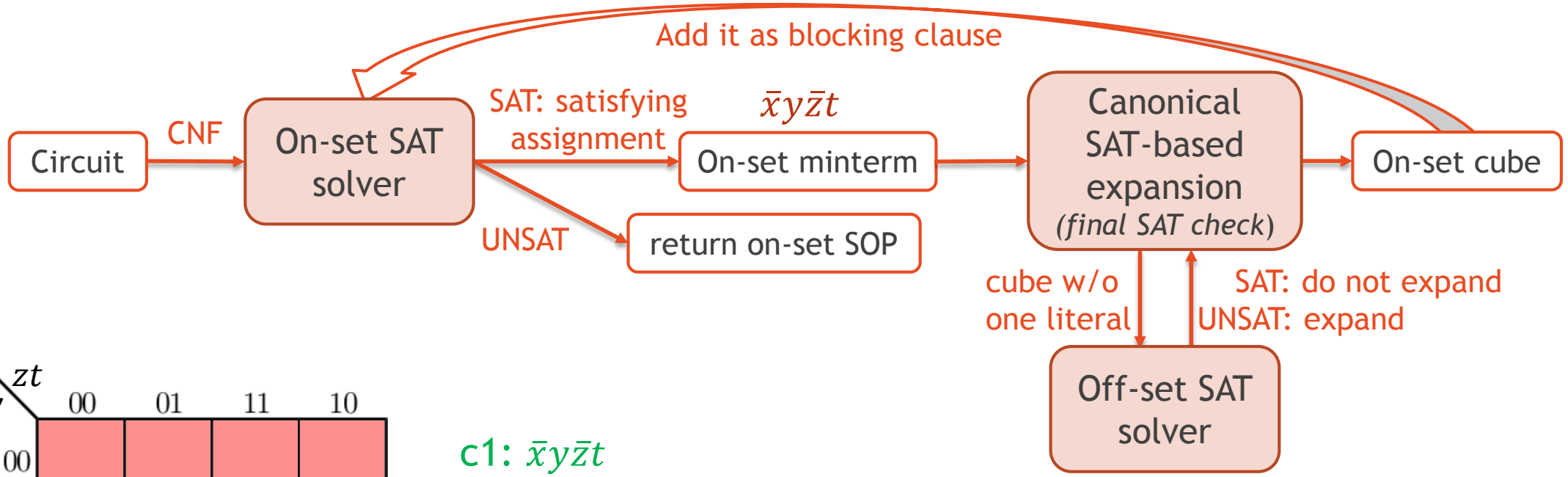


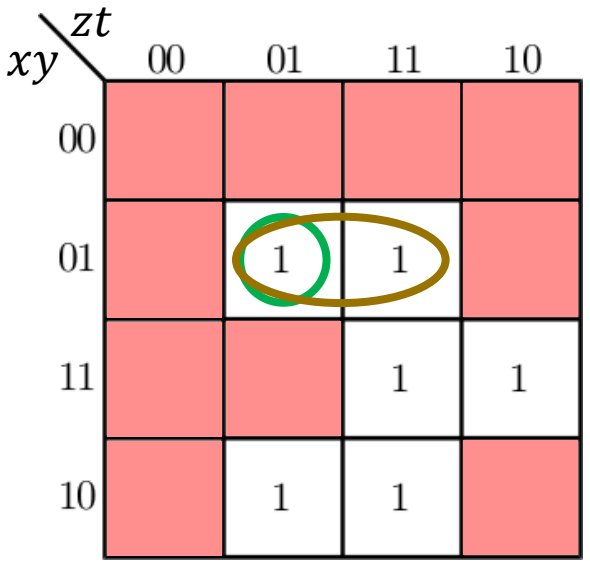
c1: $\bar{x}y\bar{z}t$

c2: $-y\bar{z}t$

c3: $\bar{x}-\bar{z}t$

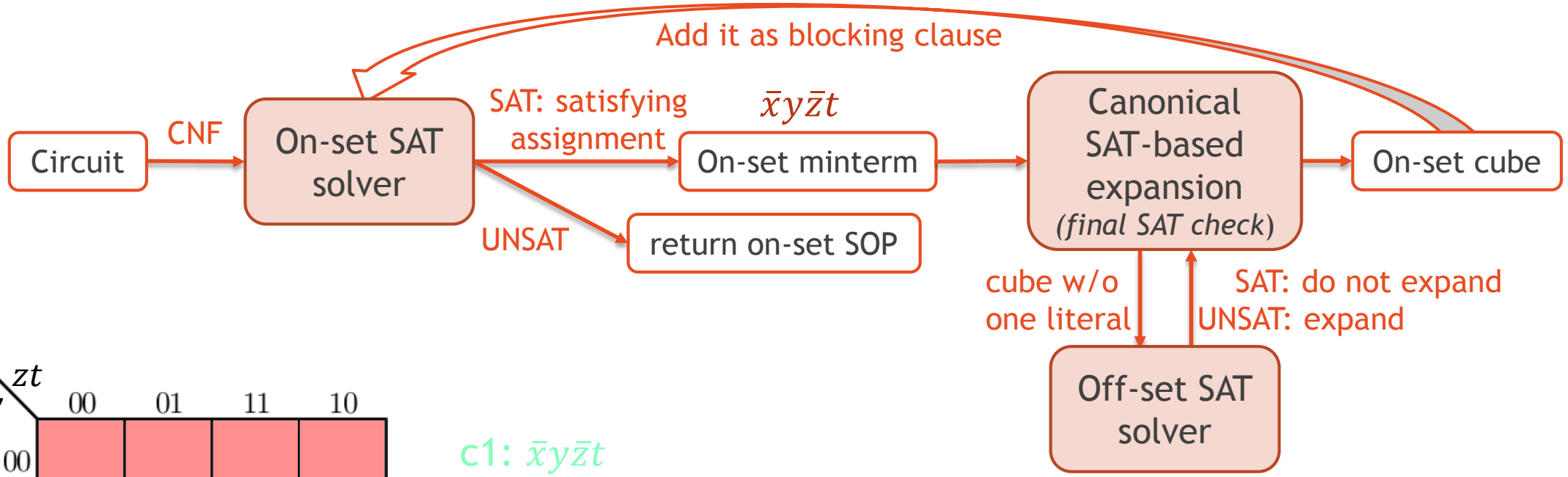
Expansion of Minterms to Cubes

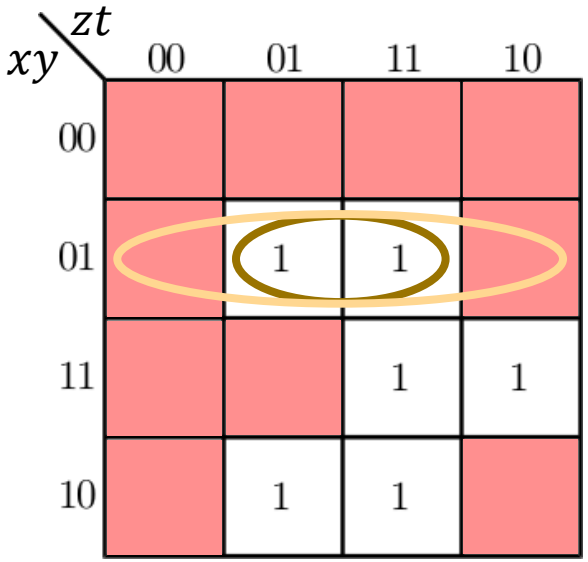




c1: $\bar{x}y\bar{z}t$
c2: $-y\bar{z}t$
c3: $\bar{x}-\bar{z}t$
c4: $\bar{x}y-t$

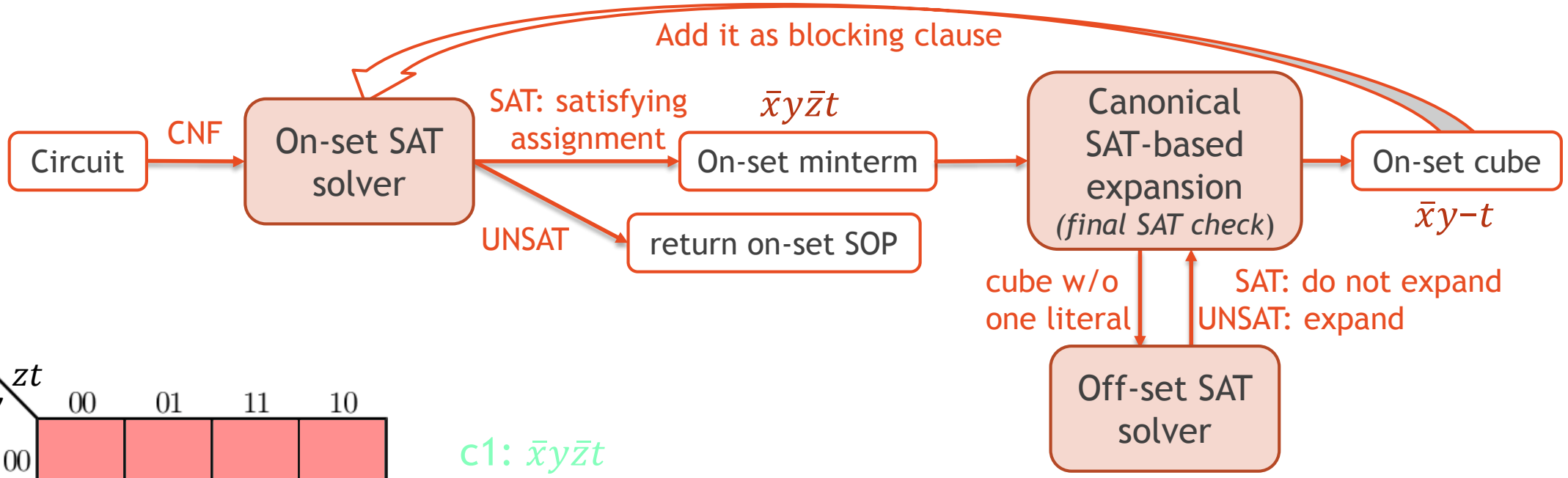
Expansion of Minterms to Cubes

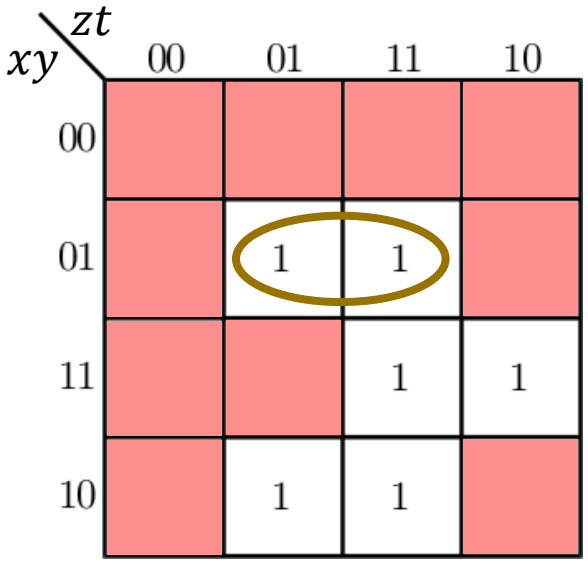




c1: $\bar{x}y\bar{z}t$
c2: $-y\bar{z}t$
c3: $\bar{x}-\bar{z}t$
c4: $\bar{x}y-t$
c5: $\bar{x}y$--

Expansion of Minterms to Cubes

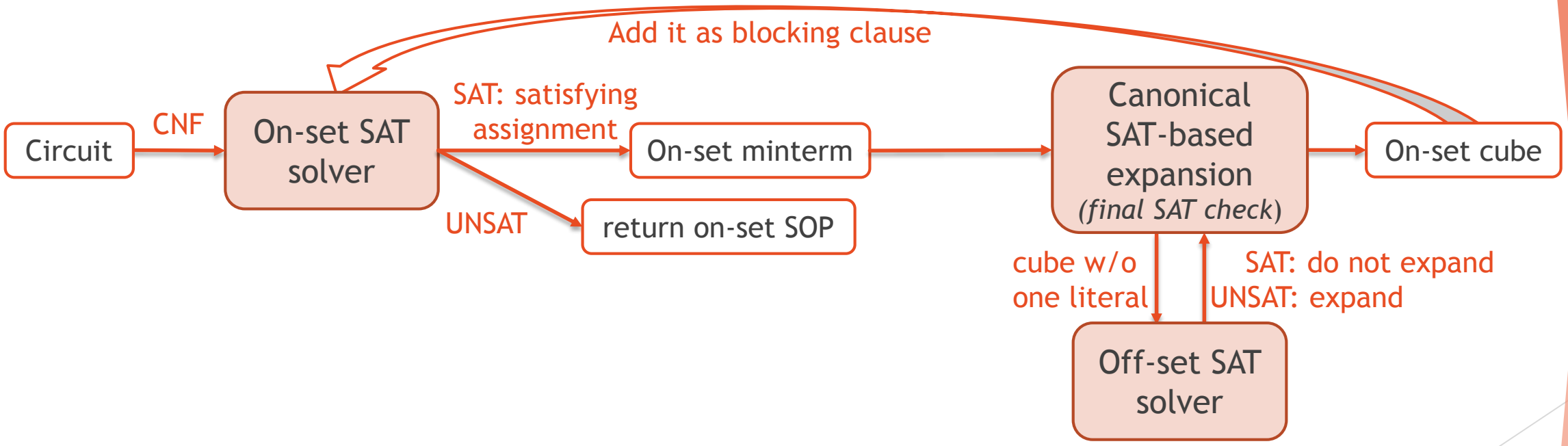




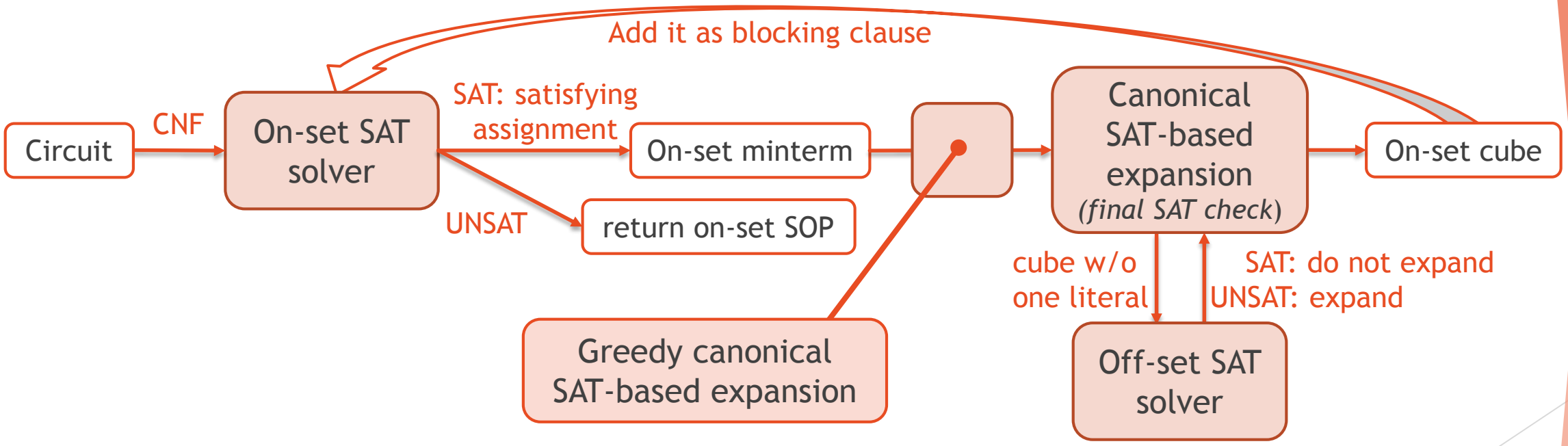
c1: $\bar{x}y\bar{z}t$
c2: $-y\bar{z}t$
c3: $\bar{x}{-}\bar{z}t$
c4: $\bar{x}y{-}t$
c5: $\bar{x}y{-}{-}$

- Canonical expansion: literals are always removed in the same order
- Prime cubes: no literal can be removed without changing the value of the function

Expansion of Minterms to Cubes

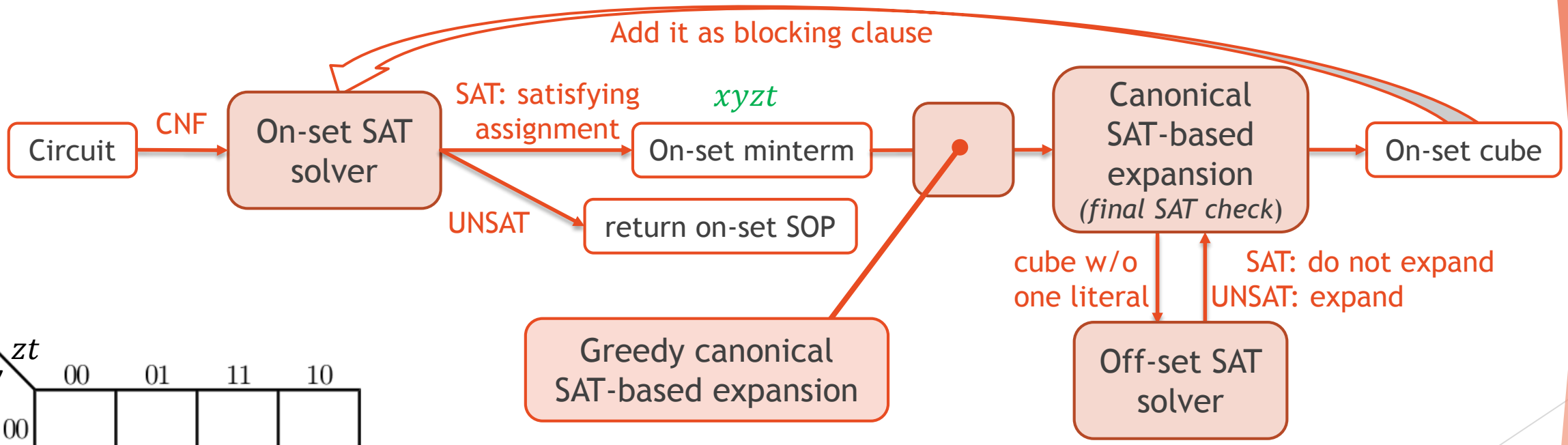


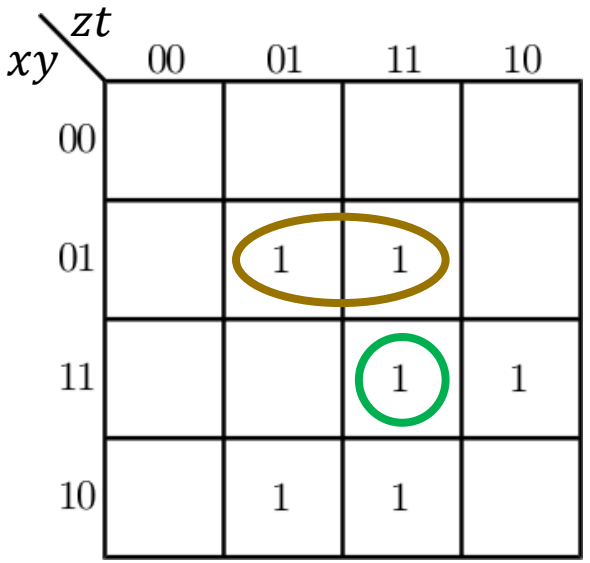
Expansion of Minterms to Cubes



- Ensure that more than one uncovered minterm is covered

Expansion of Minterms to Cubes



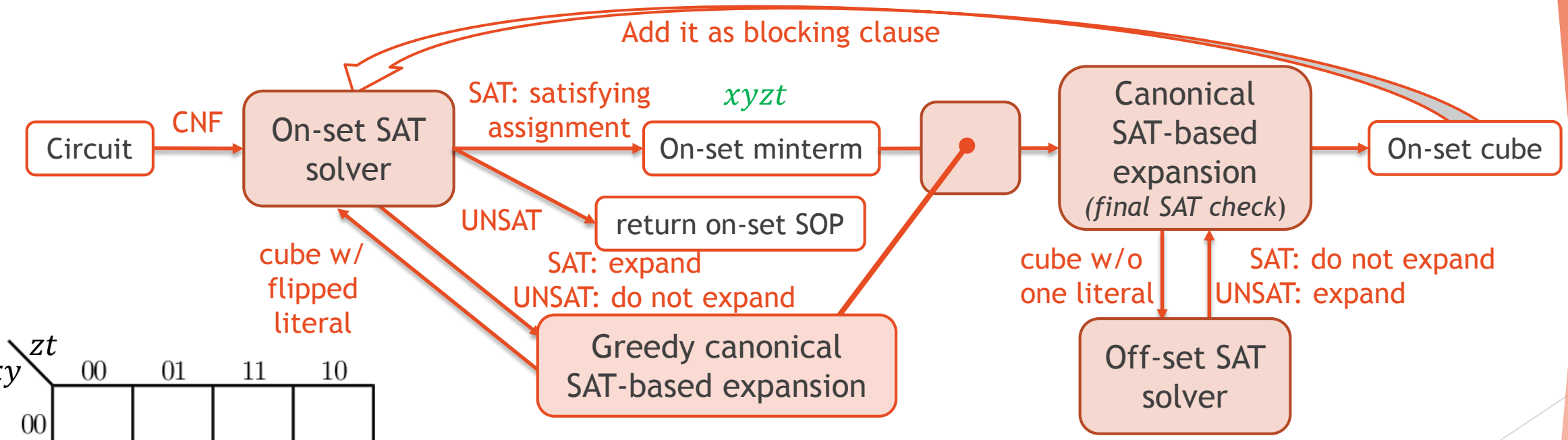


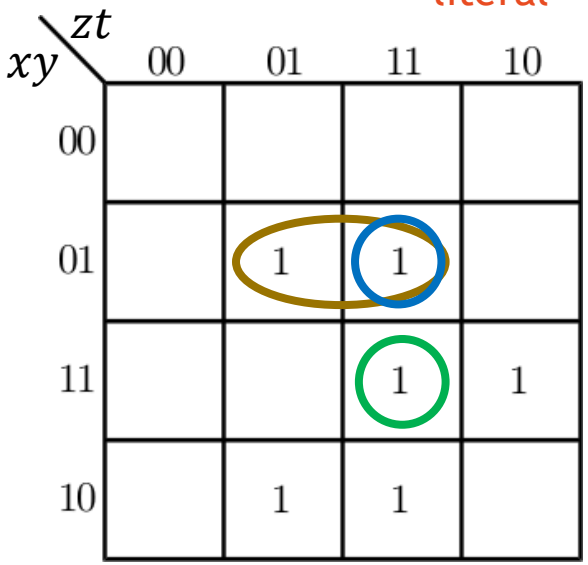
blocked: $\bar{x}y{-}t$

$xyzt$

- Ensure that more than one uncovered minterm is covered

Expansion of Minterms to Cubes





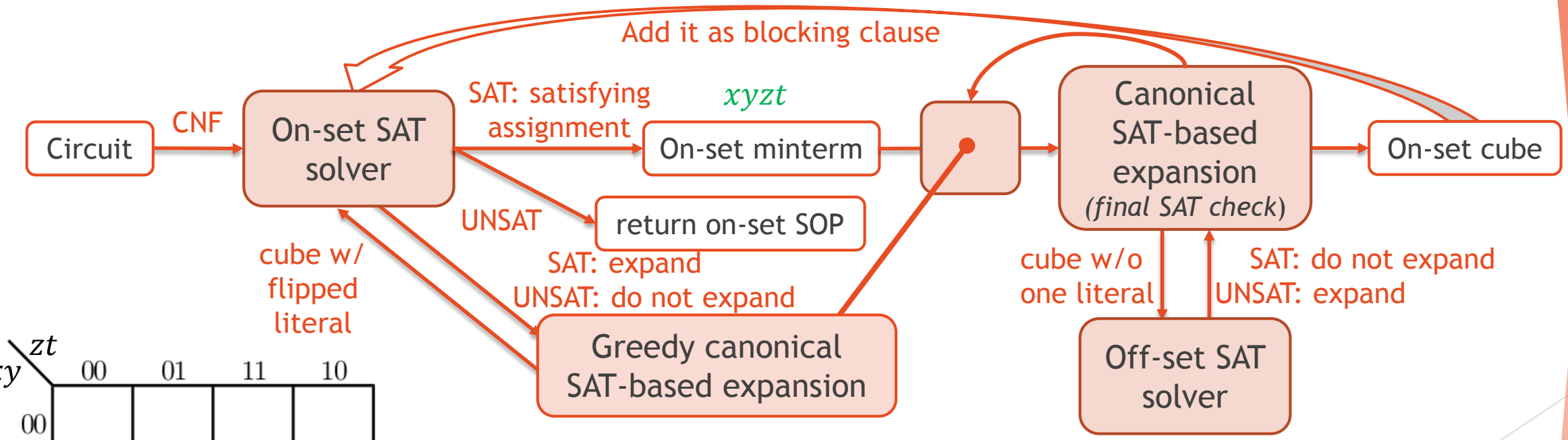
blocked: $\bar{x}y{-}t$

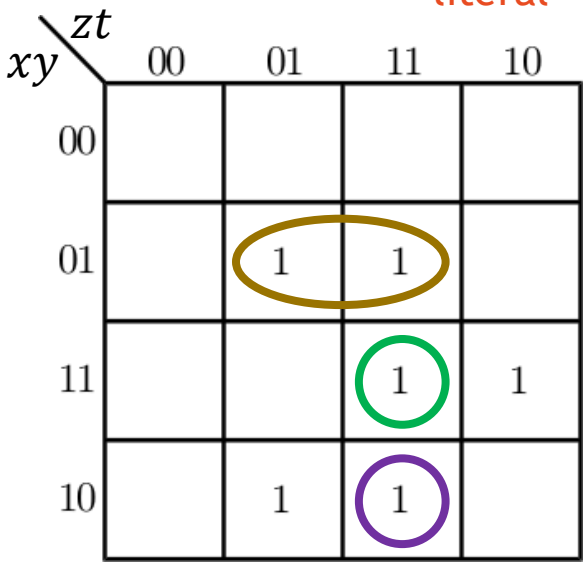
$xyzt$

$\bar{x}yzt$

- Ensure that more than one uncovered minterm is covered

Expansion of Minterms to Cubes





blocked: $\bar{x}y{-}t$

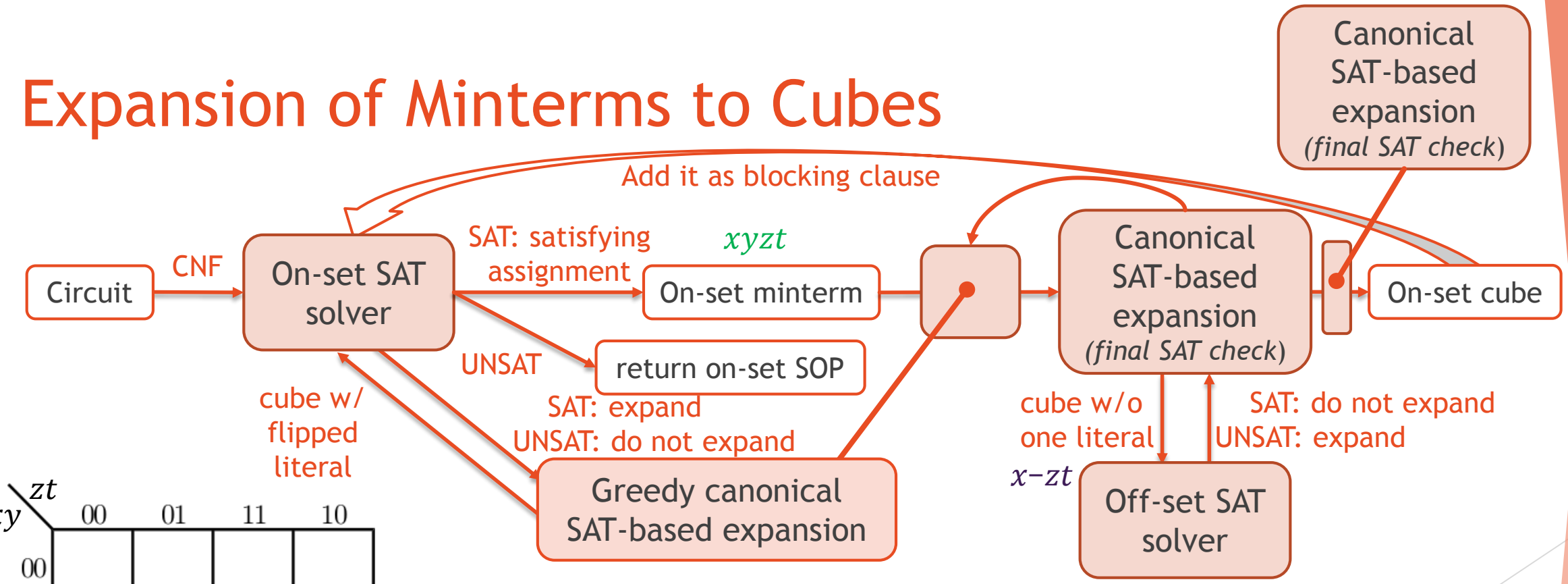
$xyzt$

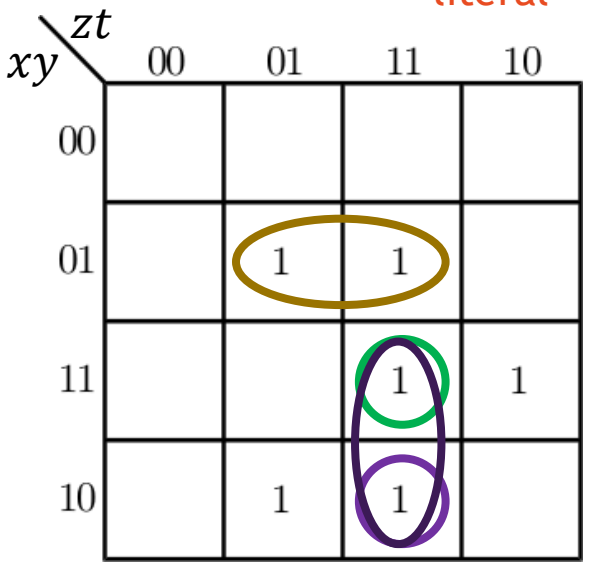
$\bar{x}yzt$

$x\bar{y}zt \rightarrow x{-}zt$

- Ensure that more than one uncovered minterm is covered

Expansion of Minterms to Cubes





blocked: $\bar{x}y-t$

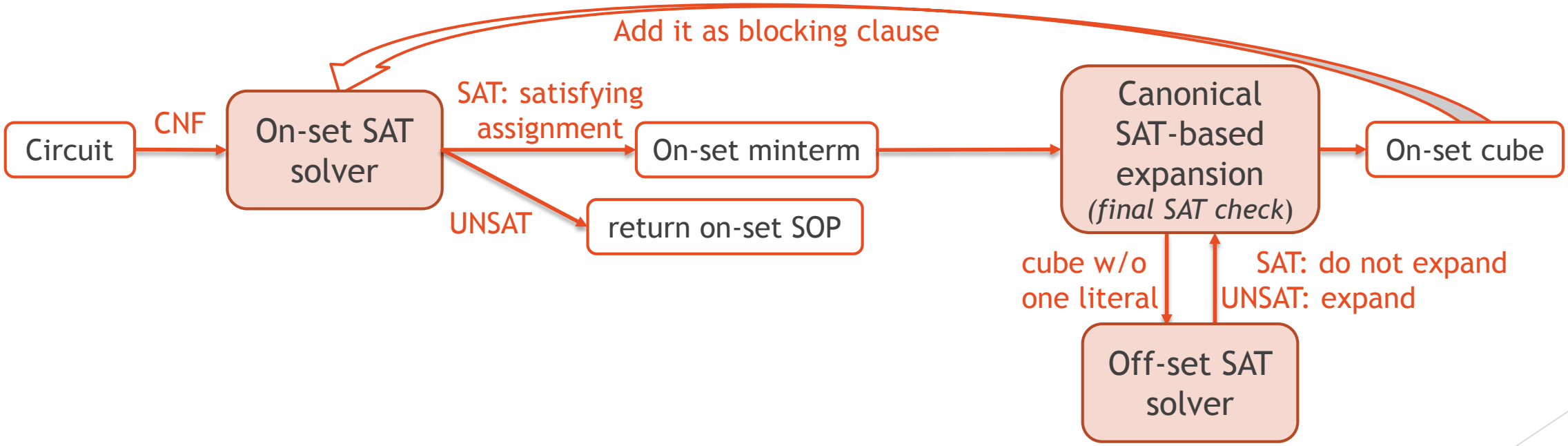
$xyzt$

$\bar{x}yzt$

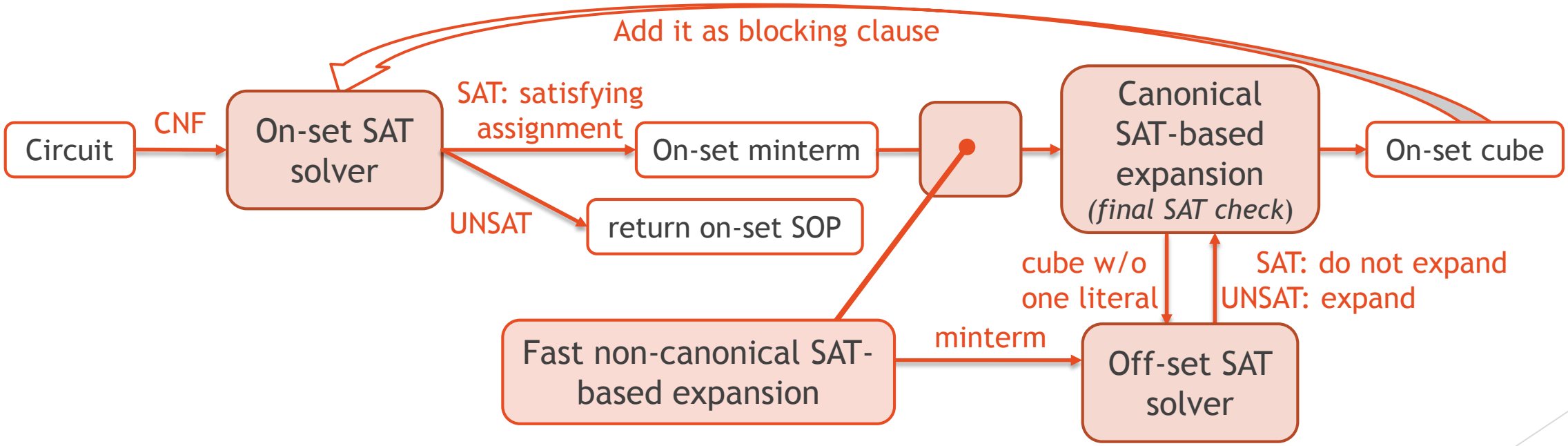
$x\bar{y}zt \rightarrow x-zt$

- Ensure that more than one uncovered minterm is covered
- Canonical expansion: literals are always removed in the same order
- At the end: iterate the final SAT check on all literals

Expansion of Minterms to Cubes

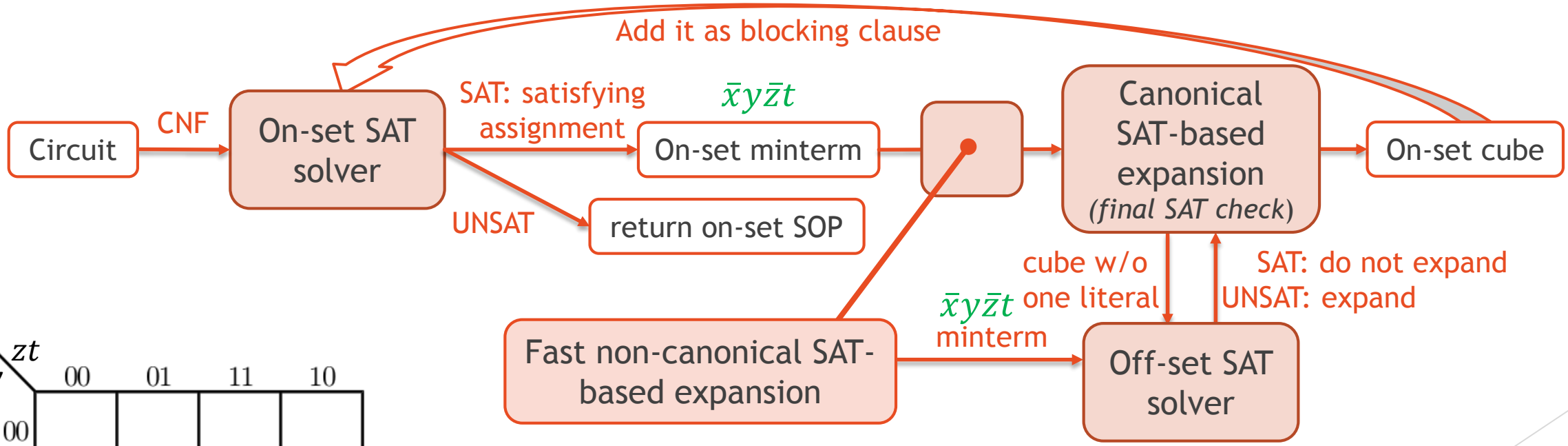


Expansion of Minterms to Cubes



- Assume the values of the minterm

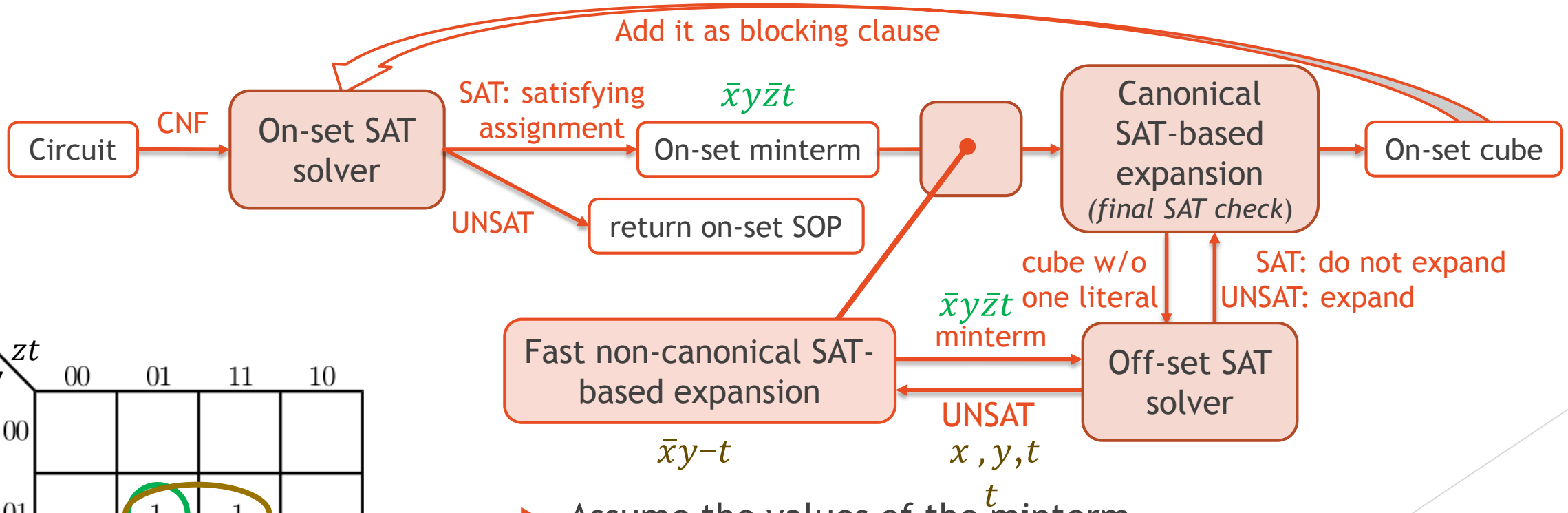
Expansion of Minterms to Cubes

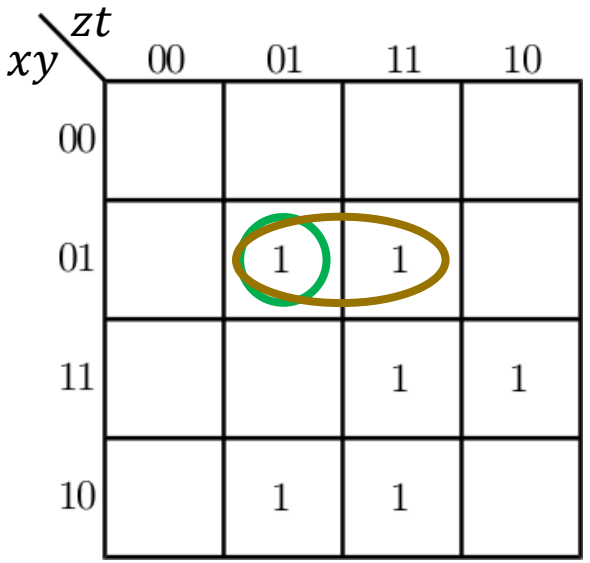


xy \ zt	00	01	11	10
00				
01		1	1	
11			1	1
10		1	1	

- Assume the values of the minterm

Expansion of Minterms to Cubes



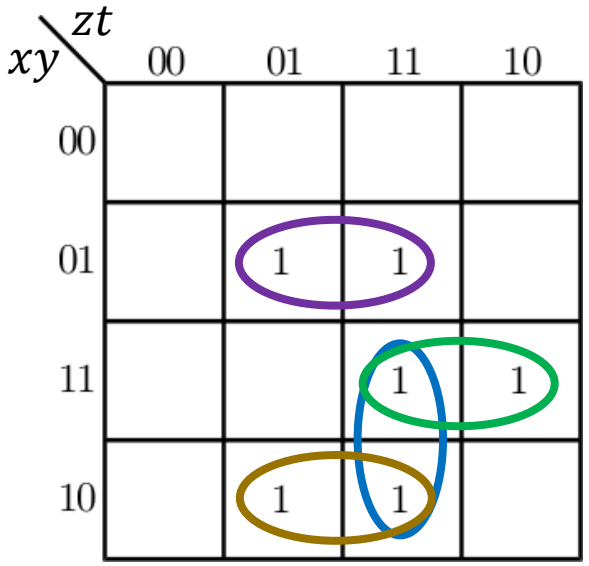


- Assume the values of the minterm
- Get the set of literals used to prove unsatisfiability (procedure “analyse final” in MiniSAT)
- Extend the literals that are not returned in the set

Outline

- SAT-based SOP Generation
  - Generation of Minterms
  - Expansion of Minterms to Cubes
  - Removing Redundant Cubes
- Methods for Runtime Improvement
- Experimental Results
- Conclusion

Removing Redundant Cubes

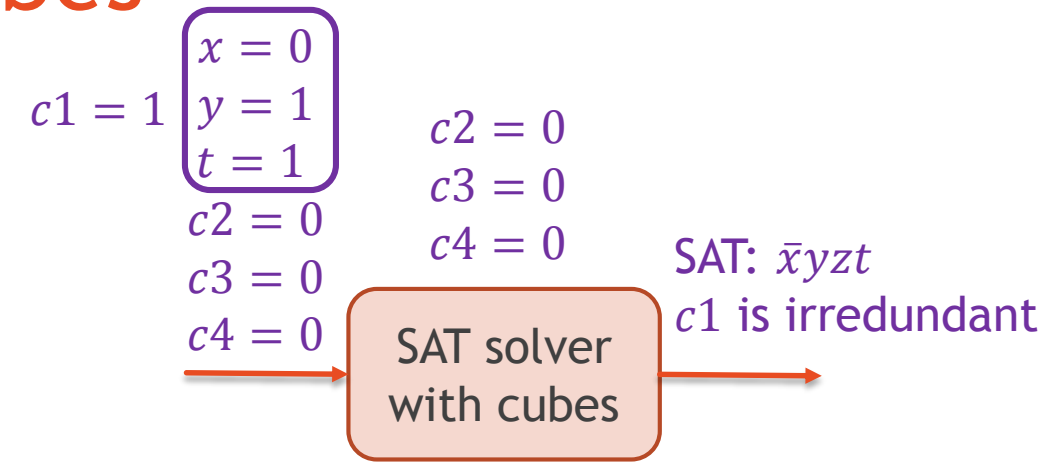


$c1 = \bar{x}yt$

$c2 = xzt$

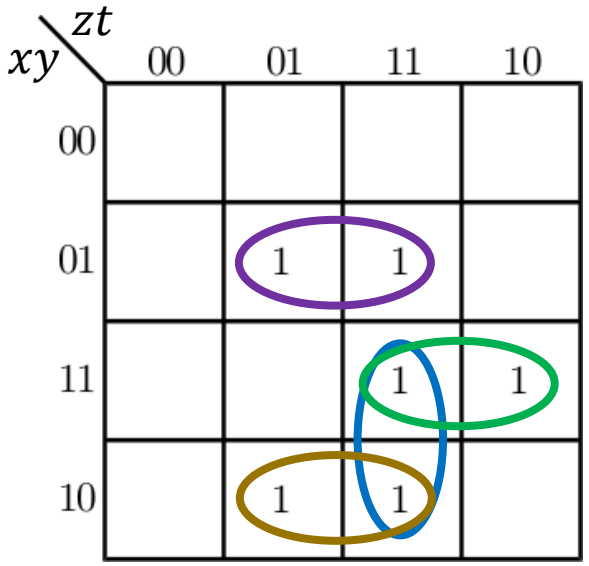
$c3 = xyz$

$c4 = x\bar{y}t$



- Initialize a SAT solver with all cubes
- Find an assignment for which $ci = 1$ and the other irredundant cubes evaluate to 0
  - SAT: ci is irredundant
  - UNSAT: ci is redundant, remove it from the SOP
- Canonical: cubes are removed always in the same order

Removing Redundant Cubes

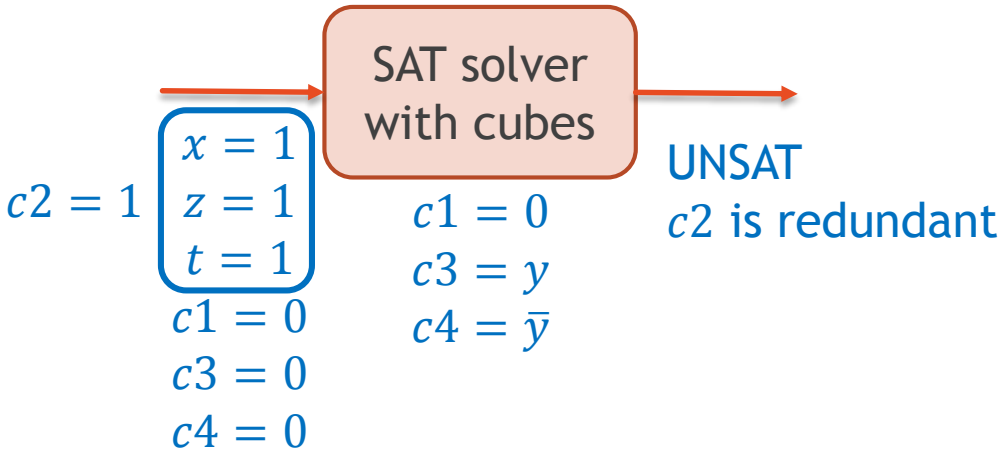


$c1 = \bar{x}yt$

$c2 = xzt$

$c3 = xyz$

$c4 = x\bar{y}t$



- Initialize a SAT solver with all cubes
- Find an assignment for which $ci = 1$ and the other irredundant cubes evaluate to 0
  - SAT: ci is irredundant
  - UNSAT: ci is redundant, remove it from the SOP
- Canonical: cubes are removed always in the same order

Outline

- SAT-based SOP Generation
  - Generation of Minterms
  - Expansion of Minterms to Cubes
  - Removing Redundant Cubes
- Methods for Runtime Improvement
- Experimental Results
- Conclusion

Methods for Runtime Improvement

- Generate simultaneously on-set and off-set SOPs

On-set SOP

x_1	x_2	x_3	f
-	1	1	1
1	1	-	1

$f = x_2x_3 + x_1x_2$

Off-set SOP

x_1	x_2	x_3	f
0	0	-	0
0	-	0	0
1	0	-	0

$\bar{f} = \bar{x}_1\,\bar{x}_2 + \bar{x}_1\,\bar{x}_3 + x_1\bar{x}_2$

Avoid generating first the larger SOP

Future work: generate each set in parallel

- Prioritize outputs with large SOP
  - Sort outputs by size of their input support
  - Generate SOPs in decreasing order

S_3: 9 inputs

S_2: 7 inputs

S_1: 5 inputs

S_0: 3 inputs

Early termination when the large SOP exceed resource limits

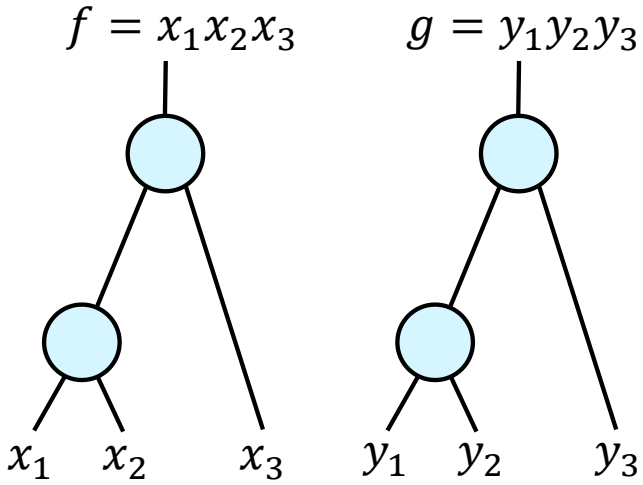
Future work: generate SOP of each output in parallel

Methods for Runtime Improvement

- Detect isomorphic outputs

  Benefit from structure sharing

  - Isomorphic functions: implement an identical function using different inputs



Isomorphic class 1
Func: f, g
Repr: $f = x_1x_2x_3$

  - Generate SOP only for one output per class (the representative)
  - Duplicate the generated SOP for the other outputs from the class

- Share one CNF among all outputs

  Benefit from logic sharing

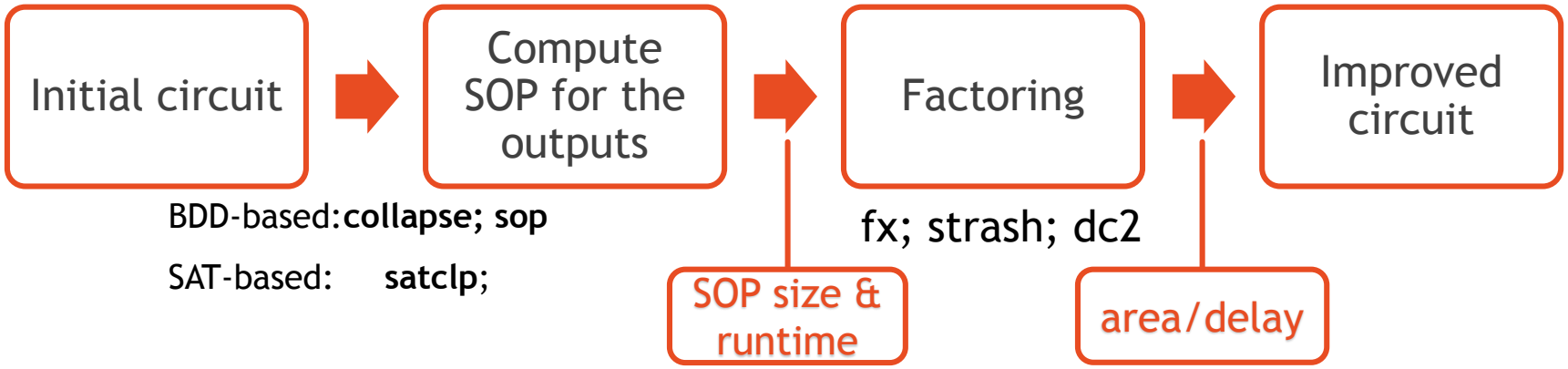
  - Generate one CNF for the complete circuit
  - For each output, initialize the SAT solver with the CNF part of the output

Outline

- SAT-based SOP Generation
  - Generation of Minterms
  - Expansion of Minterms to Cubes
  - Removing Redundant Cubes
- Methods for Runtime Improvement
- Experimental Results
- Conclusion

Experimental Setup

- Implemented the SAT-based SOP generation in ABC
- Compared flows for SOP generation

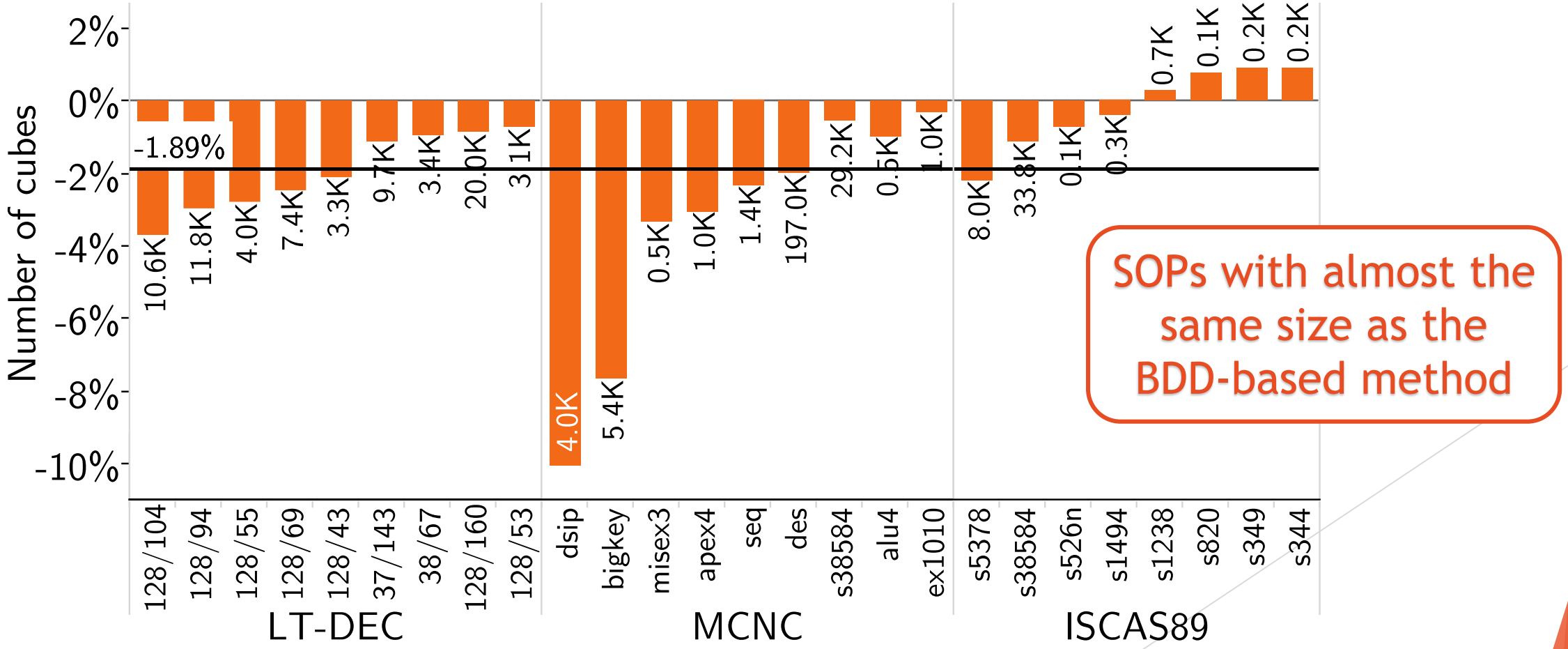


- For the BDD-based SOPs, generate 5 SOPs using different variable orders
- For the SAT-based SOPs, generate 12 SOPs using different options
  - Non-canonical or canonical SOPs
  - Order variables by fanout number
  - Reverse variable order
  - Shared CNF by primary outputs

Experimental Setup

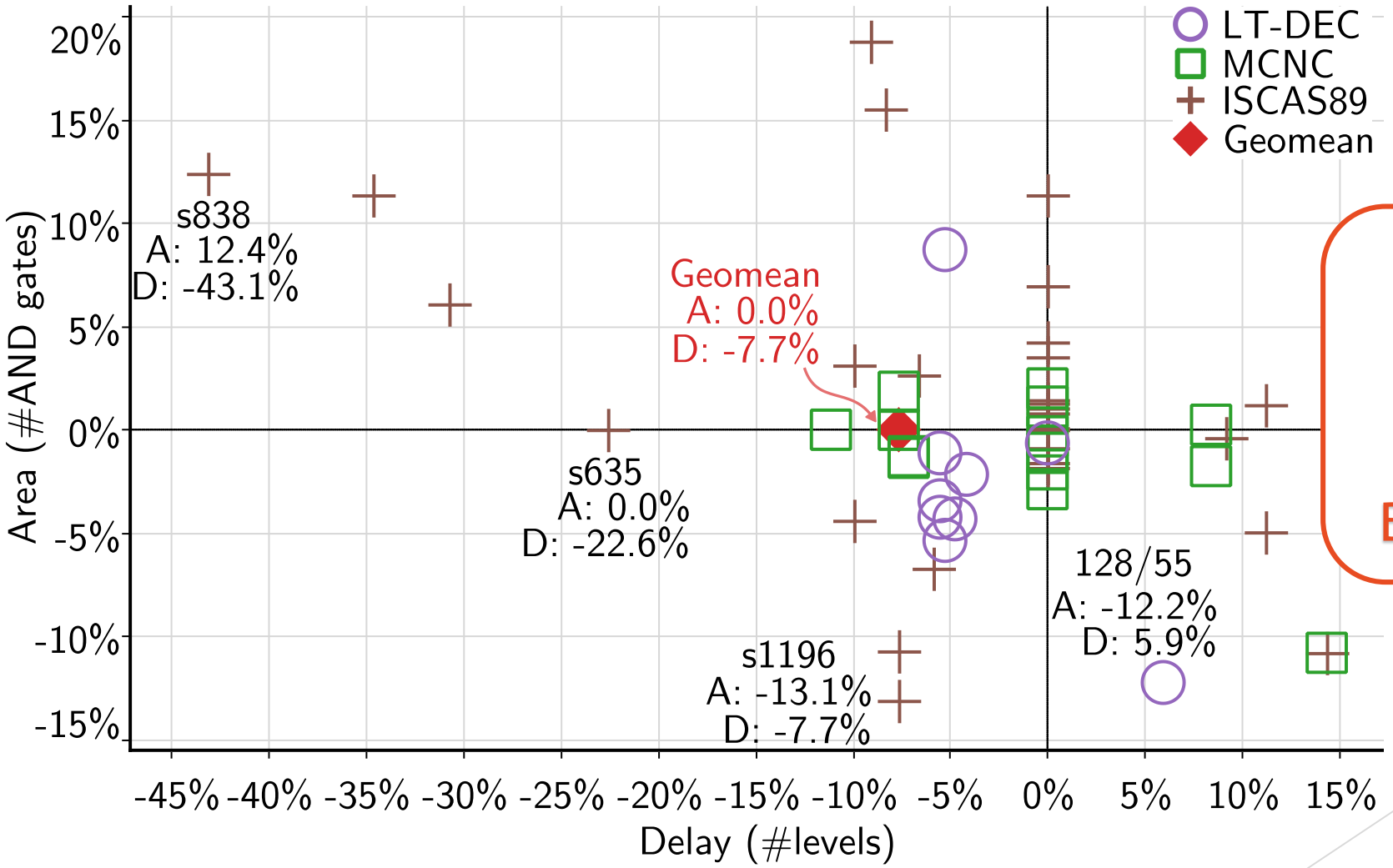
- Benchmarks
  - Set of 20 large MCNC benchmarks
  - Set of 32 ISCAS'89 benchmarks
  - Set of 9 logic tables from instruction decoder unit (LT-DEC)

Comparison of SOP Size



SOPs with almost the same size as the BDD-based method

Comparison of Area-Delay Product



Pareto optimal multi-level implementations compared to the BDD-based method

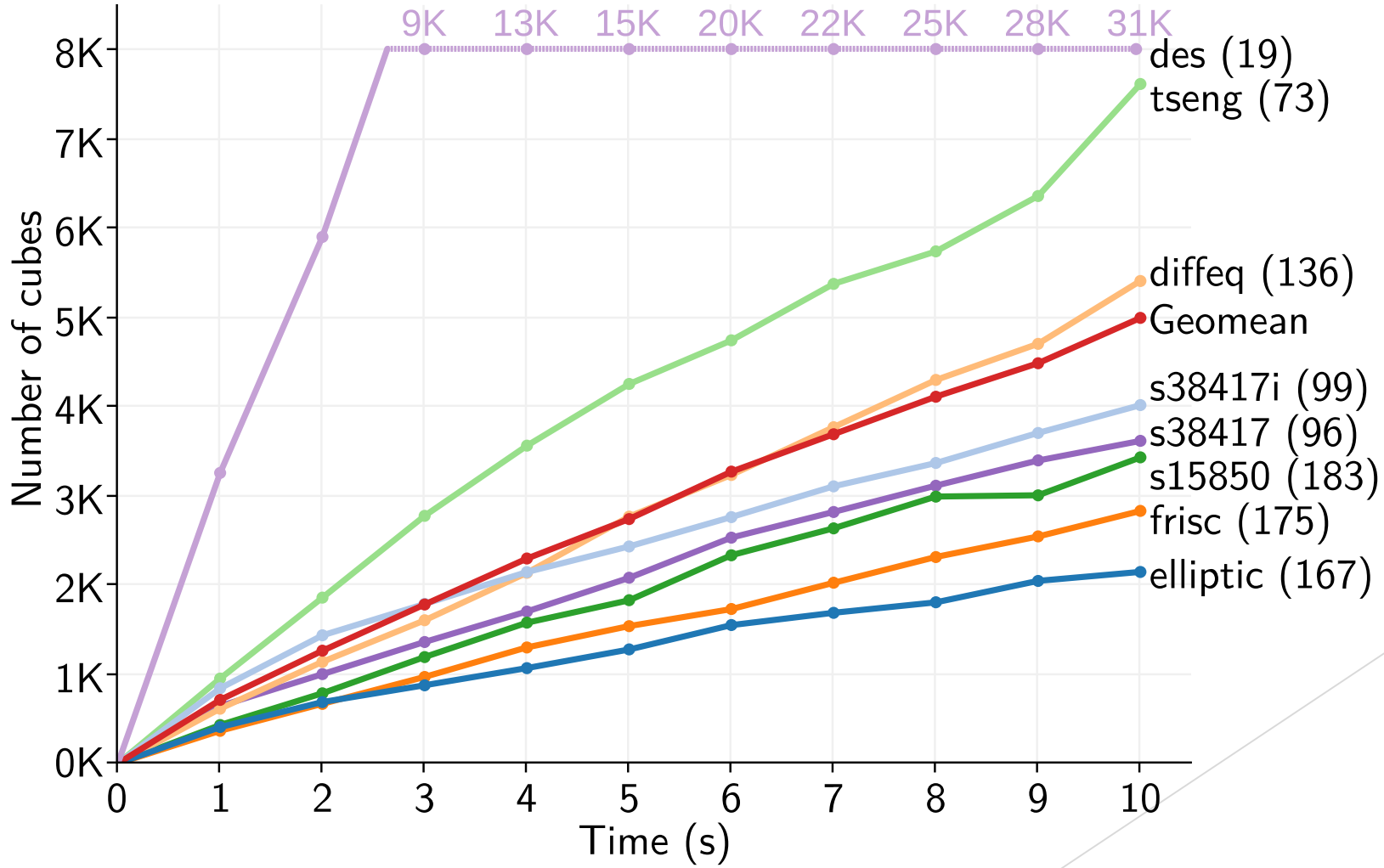
Comparison of Runtime

- Public benchmarks
  - On average, 8x slower than the BDD-based method
  - Faster for circuits with many isomorphic outputs
    - S35932 – 10 isomorphic classes for 2048 combinatorial outputs -> SAT-based: 0.06 sec; BDD-based: 1.83 sec
- Industrial benchmarks
  - Faster due to isomorphism: generated SOPs for only ~30% of the outputs
  - Hundreds of PIs/POs -> constructing the global BDD in one manager is problematic

Comparison of Runtime

- Runtime profile (for the LT-DEC benchmarks)
  - Generating minterms 8%
  - Expanding minterms to cubes 84%
  - Removing redundant cubes 2%
  - Other operations 6%

Partial SOPs



Outline

- SAT-based SOP Generation
  - Generation of Minterms
  - Expansion of Minterms to Cubes
  - Removing Redundant Cubes
- Methods for Runtime Improvement
- Experimental Results
- Conclusion

Conclusion

- We presented a complete SAT-based algorithm for **progressive generation** of **irredundant (non-)canonical SOPs**
- Quality of results
  - Decreased SOP size up to 10% compared to the BDD-based method
  - Area-delay product post factoring can be improved up to 36%
- Runtime: 8x slower on average, but faster for isomorphic circuits
- Future work
  - Improving the runtime: using parallelism, faster cube expansion
  - Dedicated SAT-based multi-output SOP computation