ASP-DAC 2007
Yokohama

Combinational Equivalence Checking Using Incremental SAT Solving, Output Ordering, and Resets

Stefan Disch

Christoph Scholl

ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Outline

- Motivation
- Preliminaries
- Our Approach
  - Output Ordering Heuristics
  - Reset Strategy
- Experiments
- Conclusion and future work

Motivation

- Combinational Equivalence Checking (CEC) is a crucial task in VLSI CAD.
- For this purpose, various methods are used:
  - BDDs
    (tend to memory explosion if monolithic BDDs are used for large circuits)
  - SAT
    (high runtime for large circuits)
  - Combination of structured methods exploiting structural similarities between the implementation and the specification.

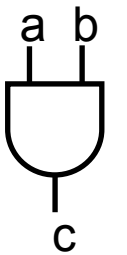
Motivation

- Here:
  - CEC using SAT for the specification and the implementation which do not show structural similarities.
- Multiple CEC problems occur, e.g., when verifying a combinational circuit with *m* outputs
- There are two traditional approaches:
  - Check each output separately
  - Check all outputs at once
- Our approach:
  Adapt incremental SAT techniques

Preliminaries:

SAT based Equivalence Checking

- Goal: Check, whether two combinational circuits are functionally equivalent or not.
- Method:
  - Connect the corresponding outputs with a miter (XOR-gate).
  - Transform this structure with a Tseitin transformation into a CNF representation.



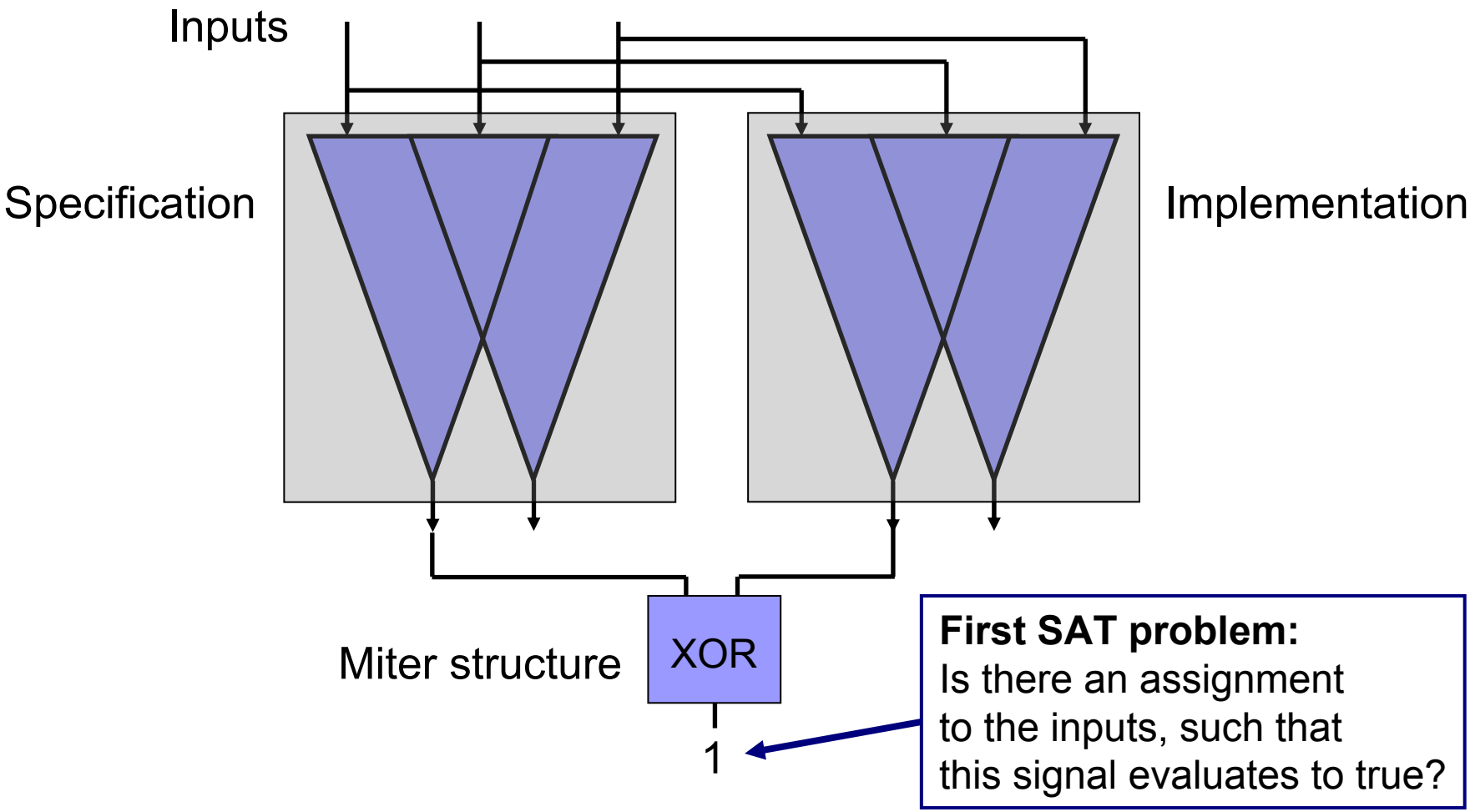
$$a \cdot b \equiv c \rightarrow (a + \overline{c})(b + \overline{c})(\overline{a} + \overline{b} + c)$$

Function which describes all consistent assignments

  - Assign the miter output variable with “true”.
  - If the resulting CNF is satisfiable, the two circuits are not equivalent.
  - Otherwise, the two circuits are equivalent.

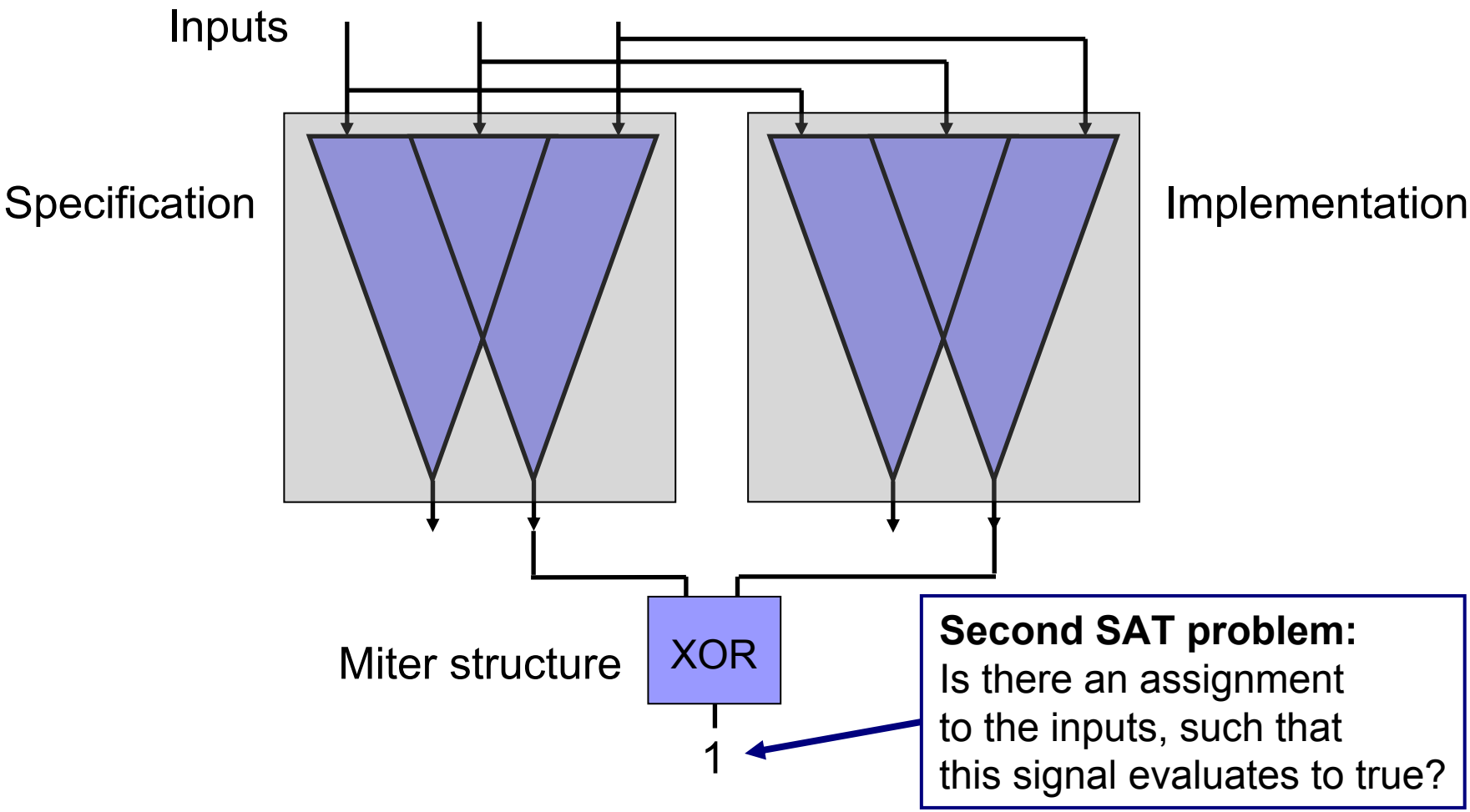
Preliminaries:

Single Output Approach (SOA)



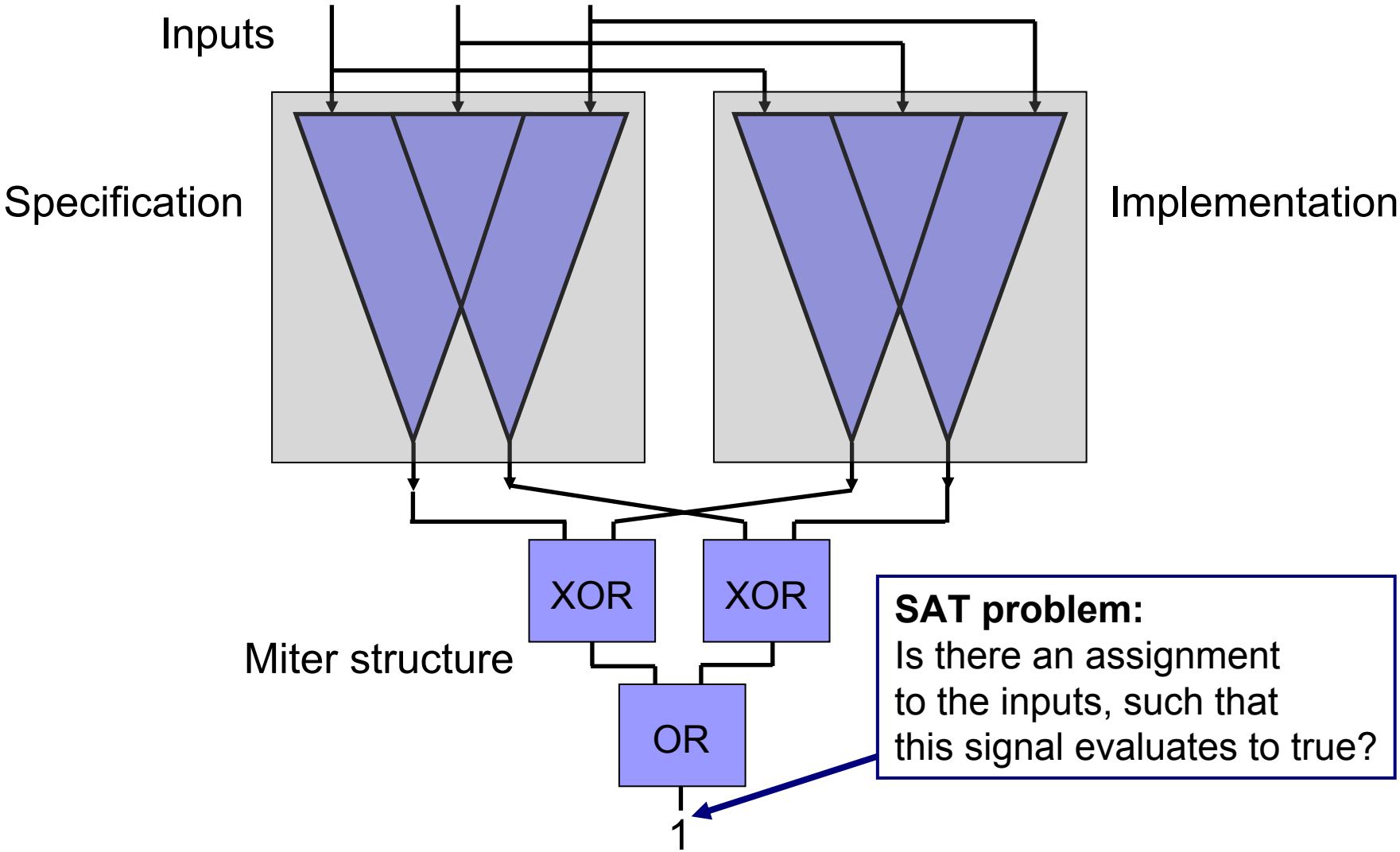
Preliminaries:

Single Output Approach (SOA)



Preliminaries:

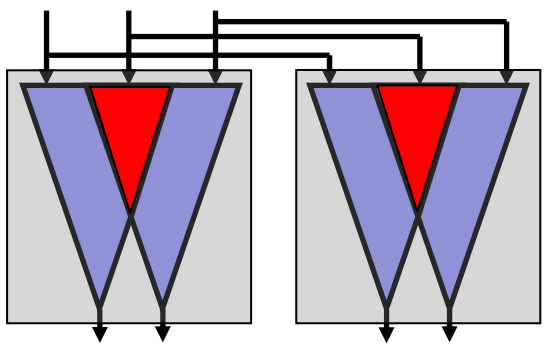
All Output Approach (AOA)



Preliminaries:

Single Output Approach (SOA)

- Pros
  - The single SAT problems are small.
  - Even if the problem is too hard for some outputs, other outputs can probably still be checked.
- Cons
  - Boolean reasoning for shared circuit structures is duplicated.



Preliminaries:

All Output Approach (AOA)

- Pros
  - Boolean reasoning for shared subcircuits is done only once.
- Cons
  - The SAT problem is larger and may be too hard to solve.

Preliminaries:

Incremental SAT

- Basic Idea:
  Reuse learned information of previous solver runs.

- After a SAT run, keep all clauses and the internal state.
- Depending on the next SAT problem:
  - Clauses no longer used and information learned from them are deleted (or deactivated), e.g. miter structure.
  - Insert additional clauses.
- Thus, learned clauses and activity values can be reused.
- This technique is mainly used in Bounded Model Checking (BMC).

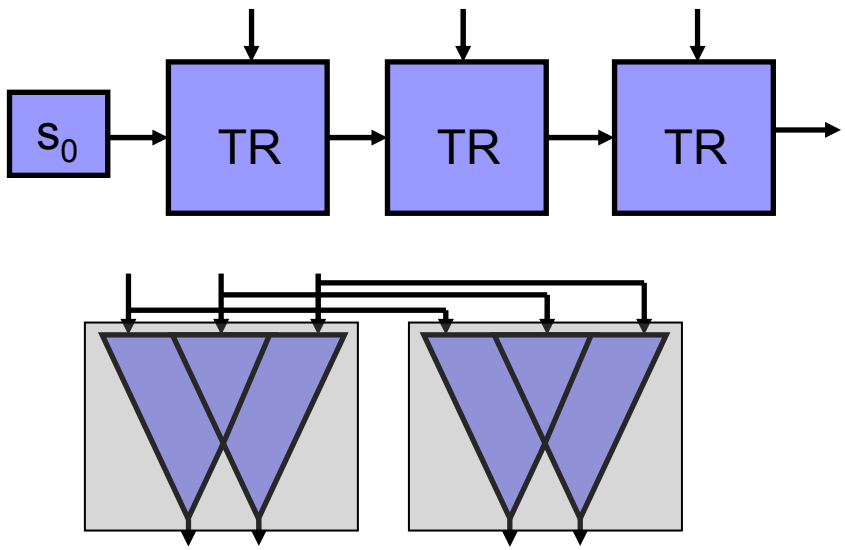
Preliminaries:

Incremental SAT

- In contrast to BMC,
  - in CEC the subsequent SAT problems often do not contain the whole structure of the previous ones.
  - the order of the SAT checks is not clear.

BMC:

CEC:



Our Approach

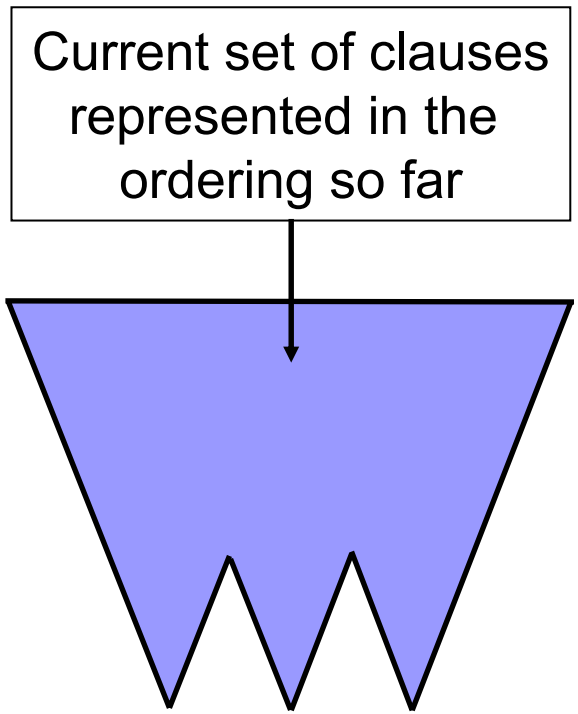
- Use a single output approach extended with incremental SAT, to reuse learned information.
- The clauses of previous runs are kept, only the miters are disabled.
- Choose a good order of the outputs.
  - ➔ Output Ordering Heuristics
- Yet, if sharing is too low, proceed with a completely new SAT instance.
  - ➔ Reset Strategy

Output Ordering Heuristics

- To support incremental SAT, a good order of the outputs is necessary.
- Idea:
  - The consecutive SAT problems should share as many clauses as possible.
  - Learned information from previous SAT calls (conflict clauses) should be reused as much as possible.
- This order is computed in a preprocessing step.
- Method:
  - Start with a miter output that has a minimal number of corresponding clauses.
  - Then, always select the miter output, that adds a minimum number of new clauses.

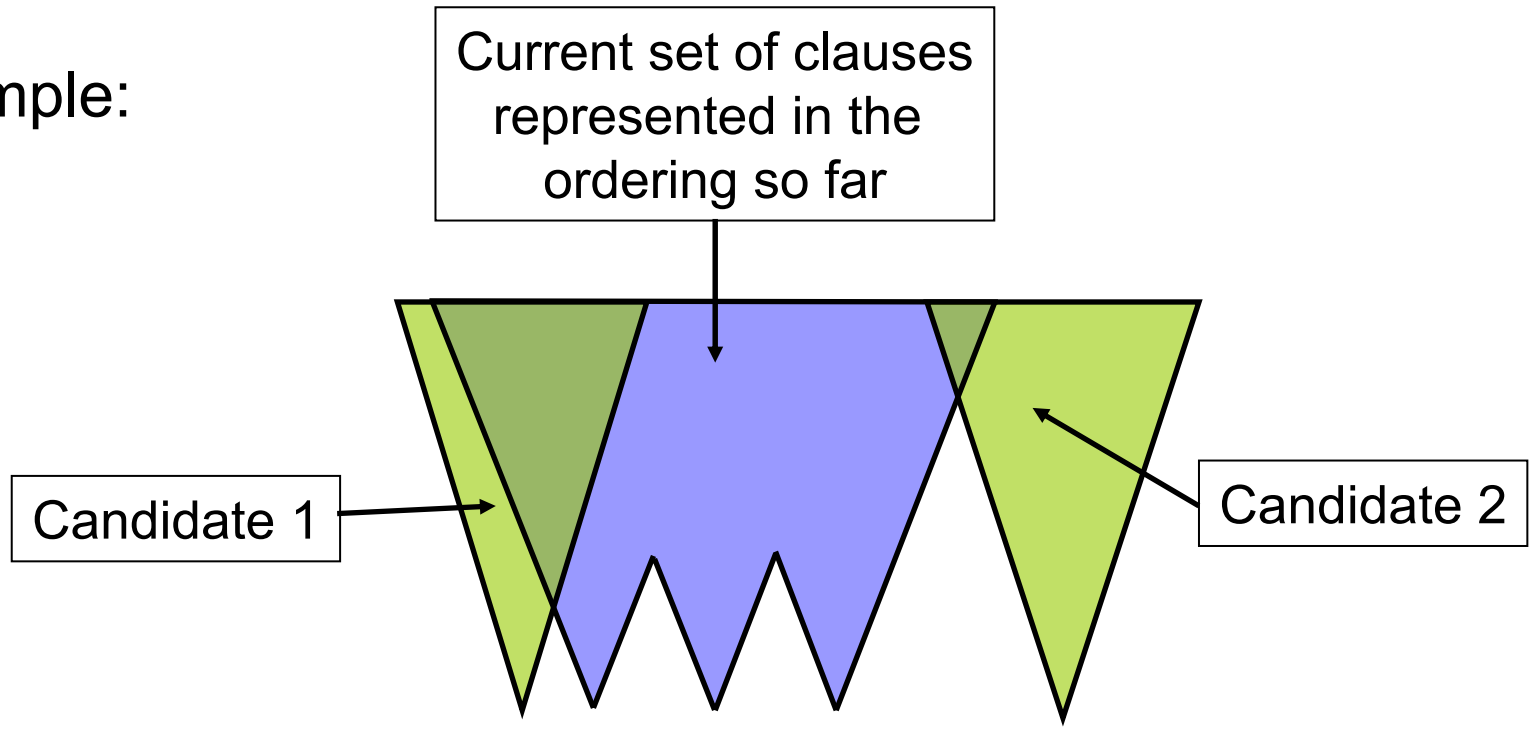
Output Ordering Heuristics

Example:



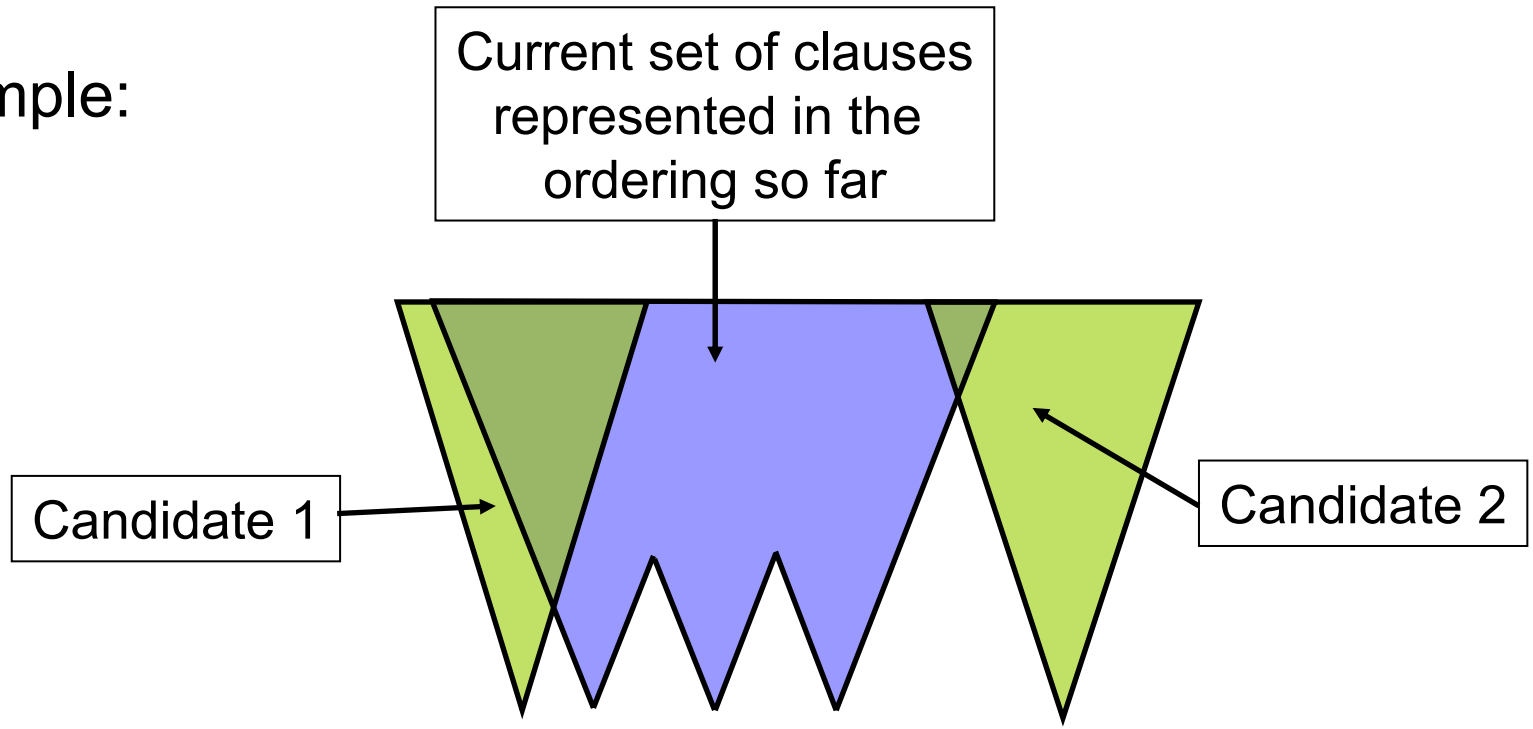
Output Ordering Heuristics

Example:



Output Ordering Heuristics

Example:

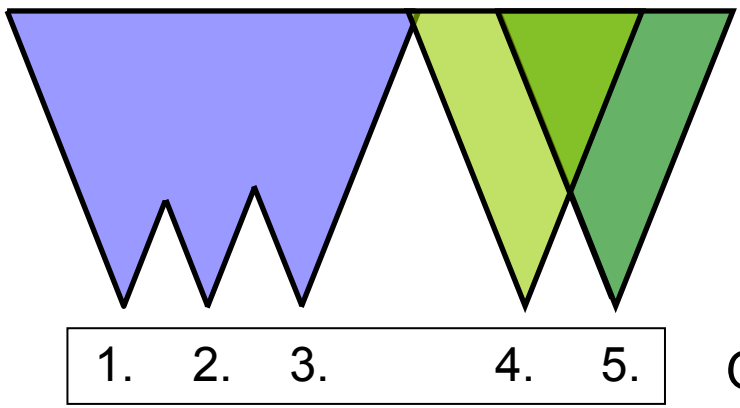


→ Select candidate 1

Reset Strategy

- If too few clauses are shared, incremental SAT has too much overhead.

Example:

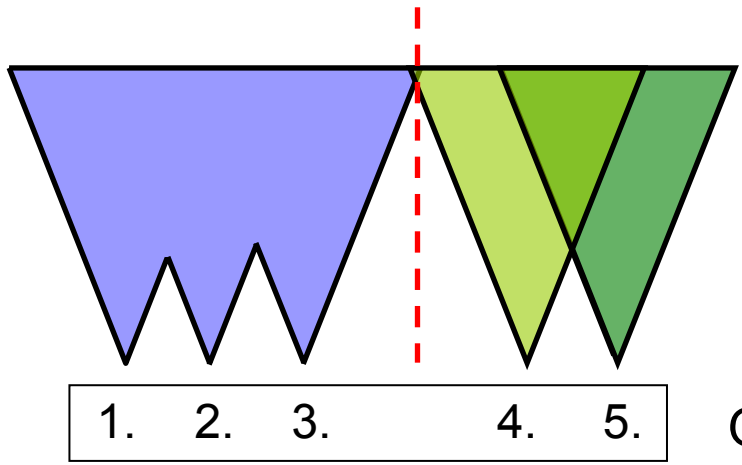


Order of the miters

Reset Strategy

- If too few clauses are shared, incremental SAT has too much overhead.

Example:

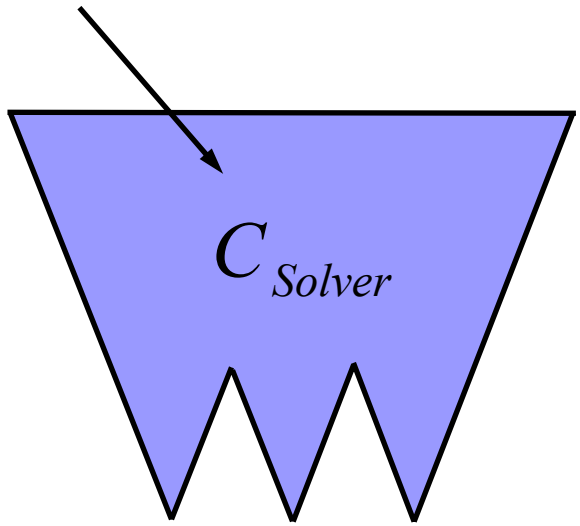


Order of the miters

- Solution:
  Drop current SAT instance and start a new one with output 4.

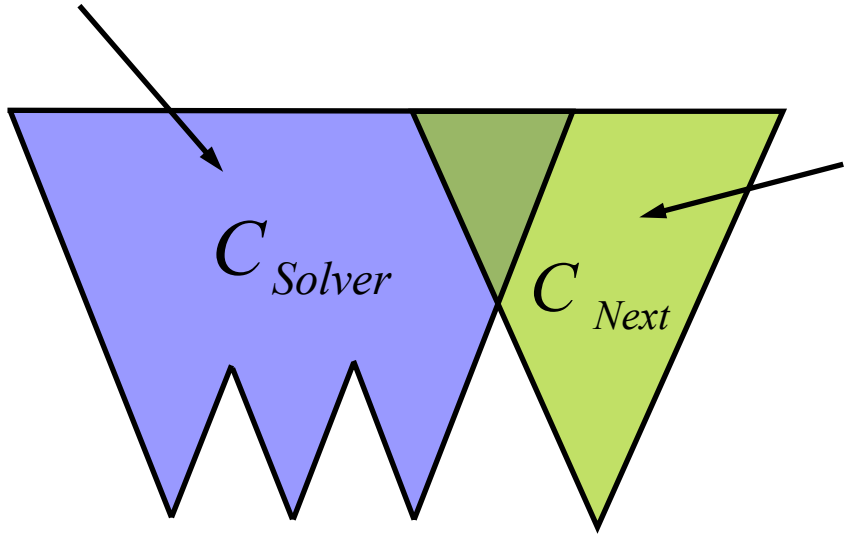
Reset Strategy

Set of clauses represented
in the current SAT instance



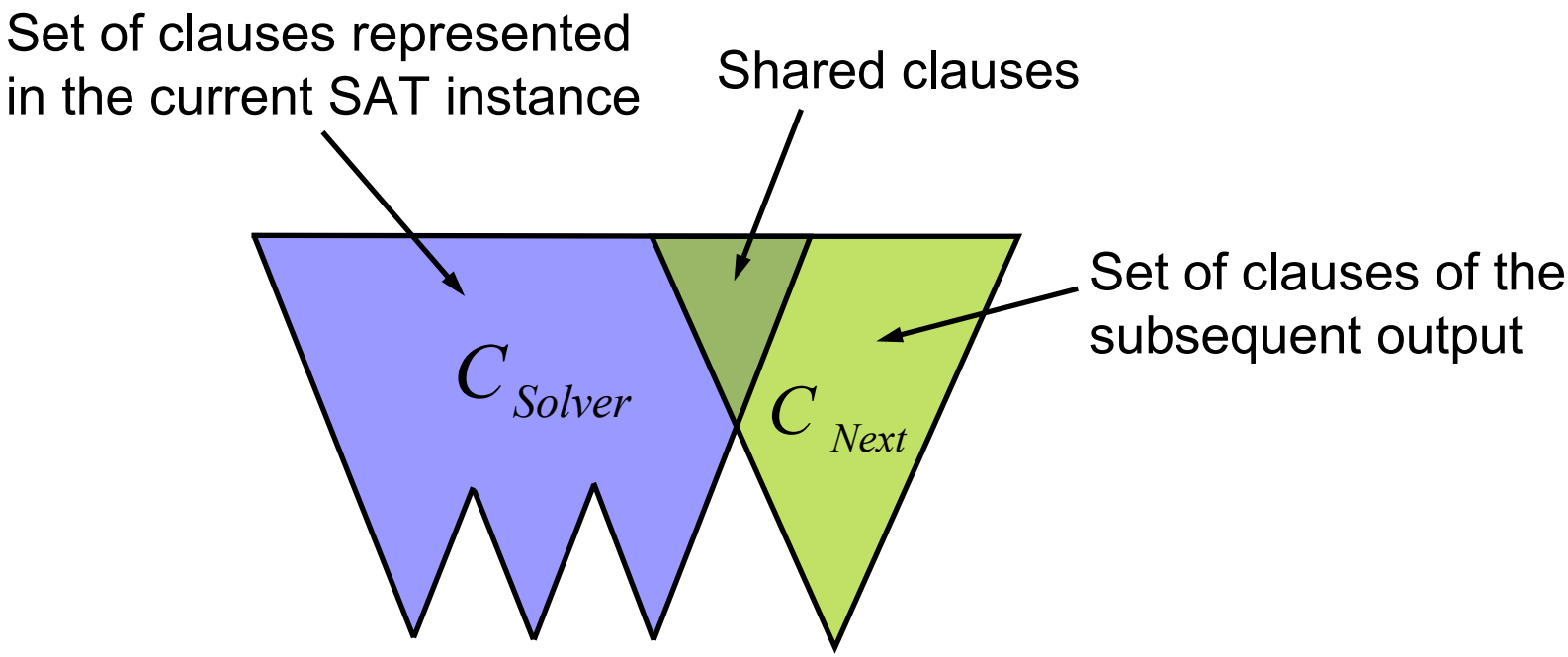
Reset Strategy

Set of clauses represented
in the current SAT instance

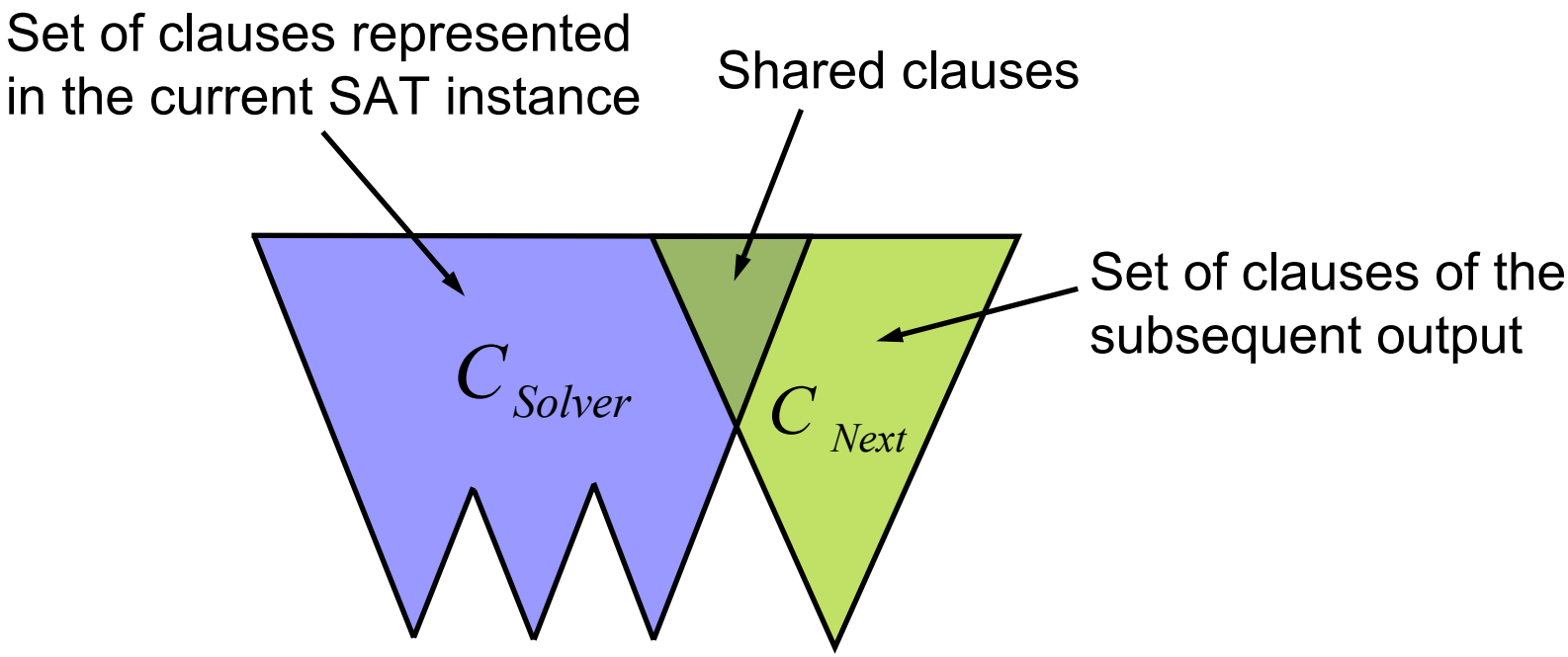


Set of clauses of the
subsequent output

Reset Strategy



Reset Strategy



if $\frac{|C_{Next} \cap C_{Solver}|}{|C_{Solver}|} < t$ (fixed ratio): perform a reset

Overall Framework

```
CEC( Circuit A, Circuit B )
{
  Circuit C = add_miters( A, B);

  order_outputs( C );

  for( int i; i <= C.nr_outputs(); ++i )
  {
     if( threshold_exceeded( C[i] )
         reset_solver();

     add_clauses( C[i] );
     assign_miter( C[i], true );
     result := start_solve();
     if( result == SAT )
        return FAILURE_FOUND;
     delete_miter_assignment( C[i] );
  }
  return NO_FAILURE_FOUND;
}
```

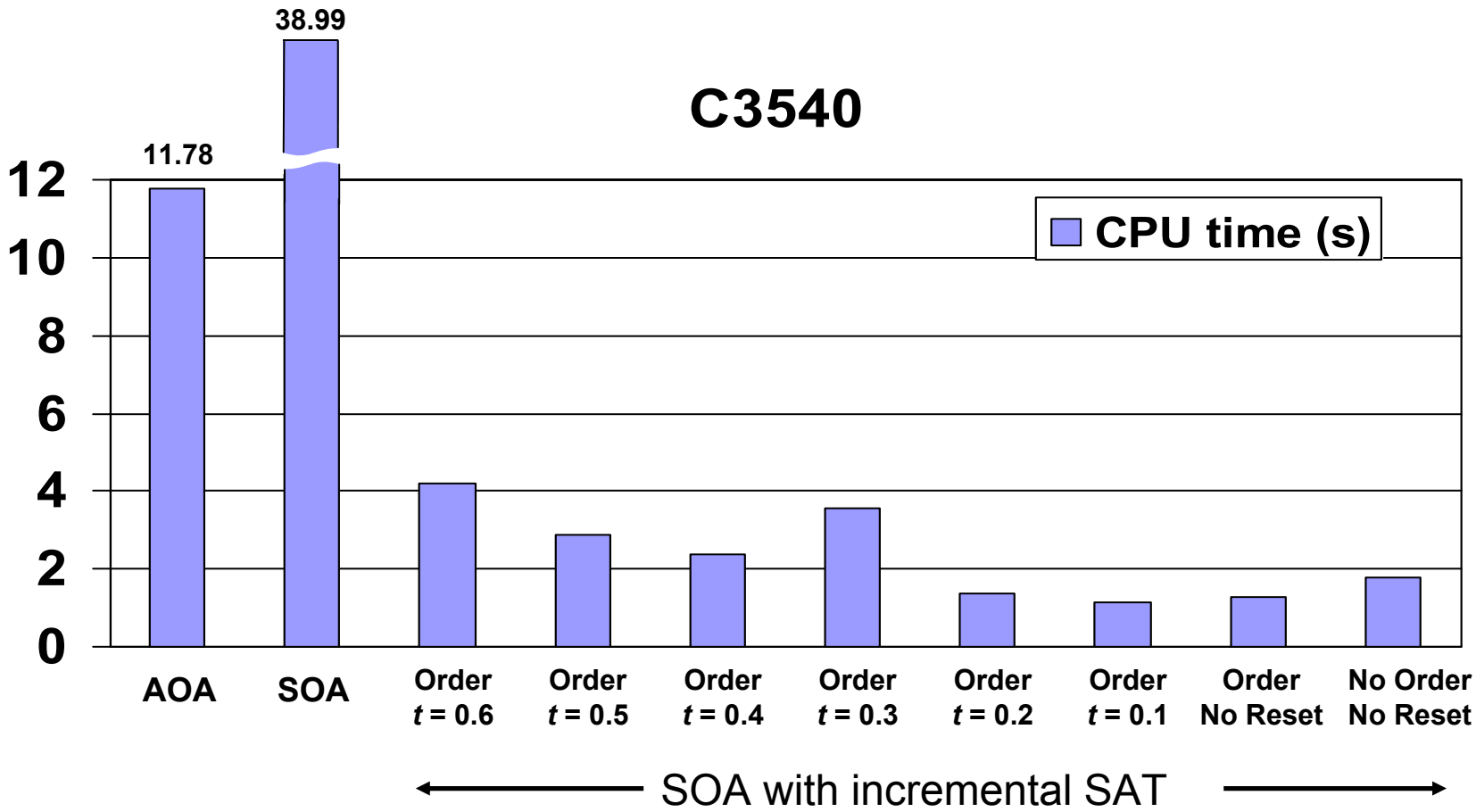
Calculate the order of the miter outputs

Check, whether to reset or not

Experimental Results

- Benchmark sets:
  ISCAS85 and ITC99 (Torino subset)
- The two circuits of each experiment were two different representations of the same benchmark.
- We used only equivalent benchmarks.
- The SAT solver MiniSAT 1.14 was used

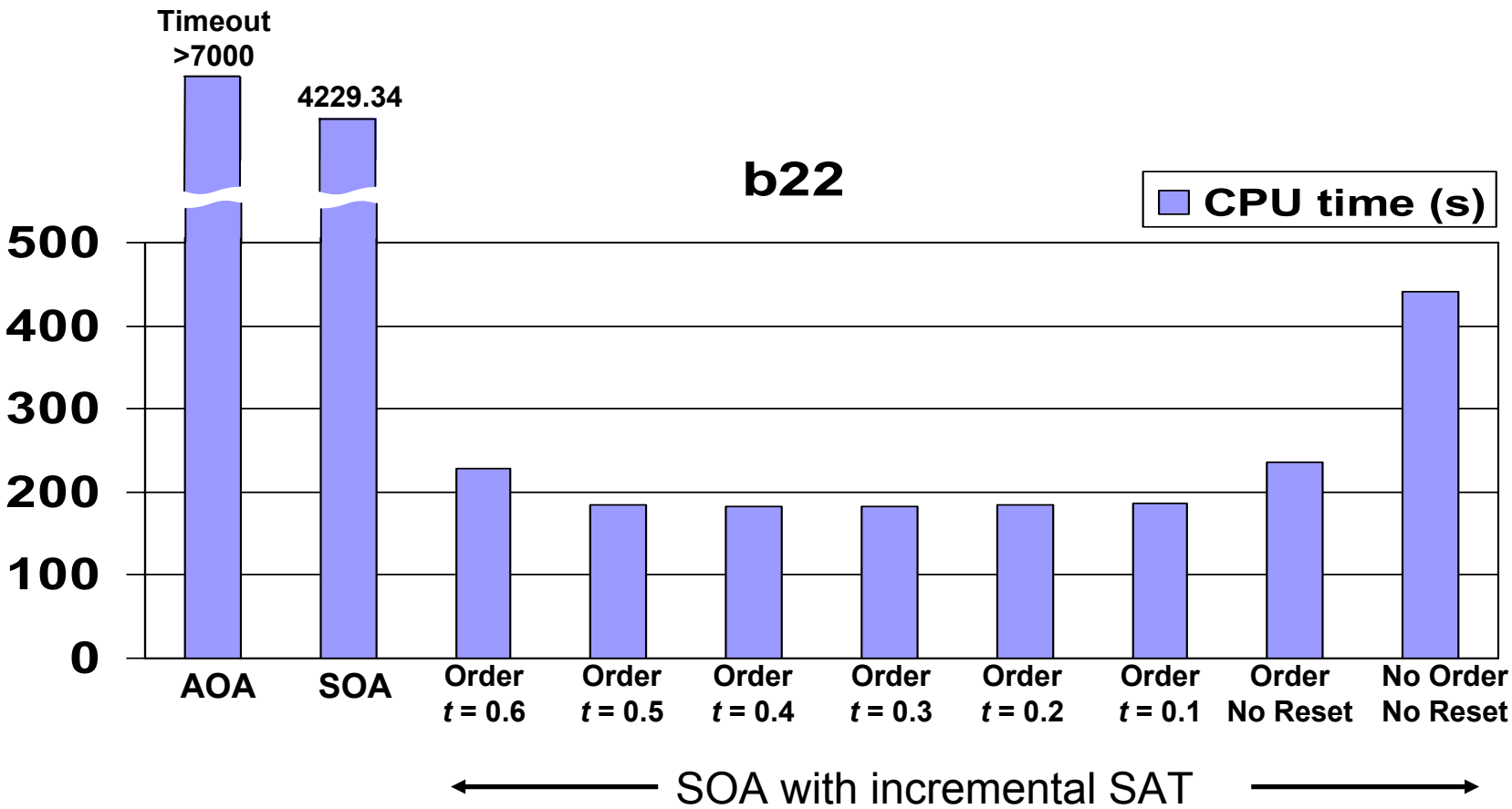
Experimental Results



t : Threshold of the reset strategy

Part of ISCAS85

Experimental Results

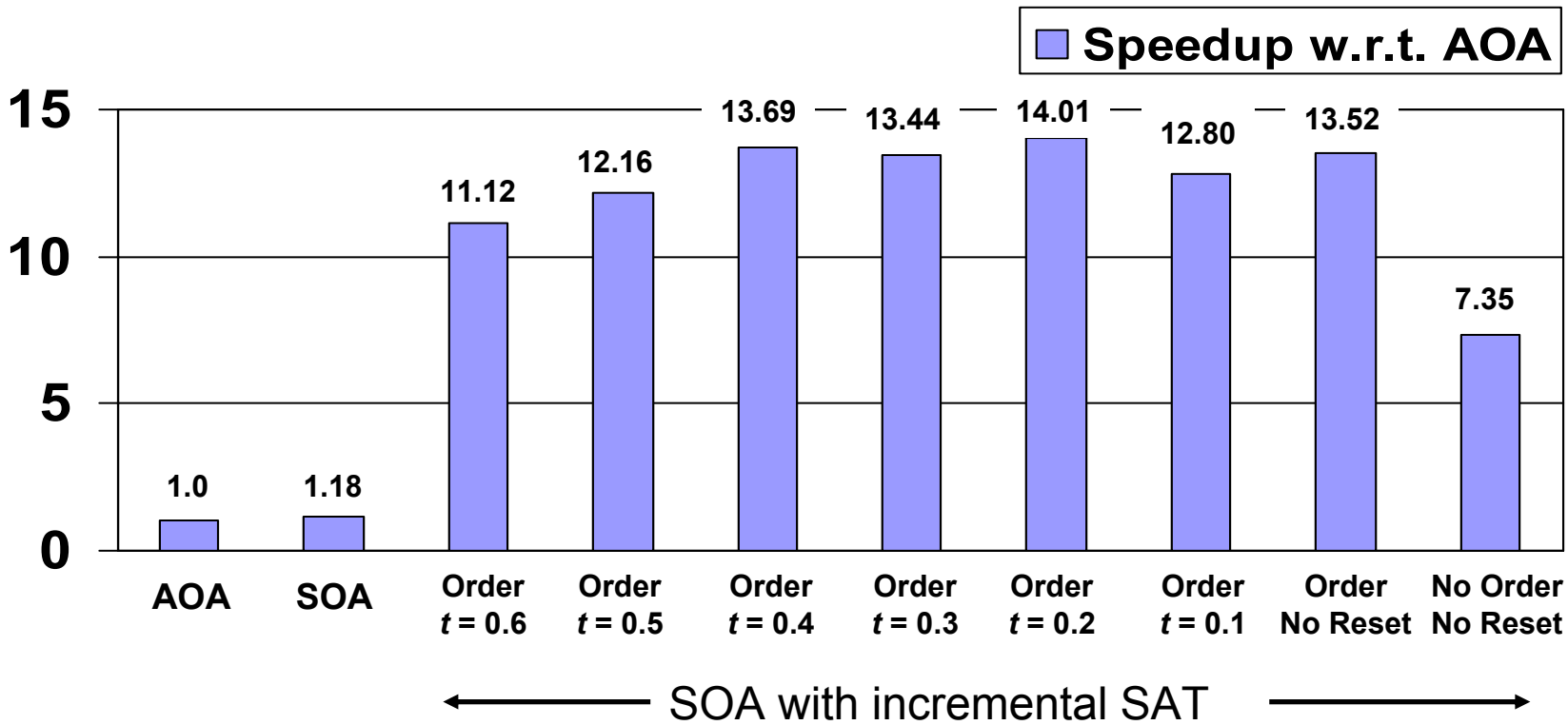


t : Threshold of the reset strategy

Part of ITC99

Experimental Results

Average Speedup over all experiments



t : Threshold of the reset strategy

Conclusion and future work

- Using incremental SAT we get significant speedups compared to the AOA and the SOA.
- Ordering and reset heuristics are important to guide the usage of incremental SAT and improving runtime furthermore.

Future work:

- Further improvement of the heuristics.
  - e.g. tighter integration with statistics collected in SAT solver.
- Generalize this approach to the context of AND-Inverter-Graph based CEC.
  - Use incremental SAT with output ordering and resets during SAT sweeping.

Thank You !