Incremental Preprocessing Methods for Use in BMC

S. Kupferschmid, M. Lewis, T. Schubert and B. Becker
{skupfers,lewis,schubert,becker}@informatik.uni-freiburg.de

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

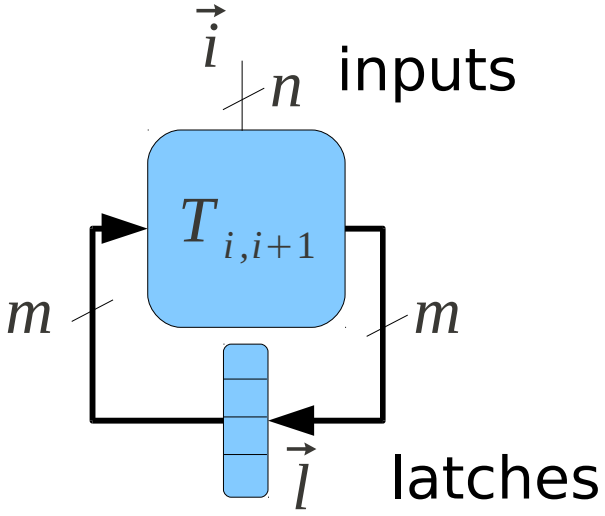
Outline

- BMC, Craig Interpolation
- Accelerating SAT-Based BMC
- Our Approach
- Results
- Conclusion

Bounded Model Checking (BMC)

- We use BMC to verify safety properties

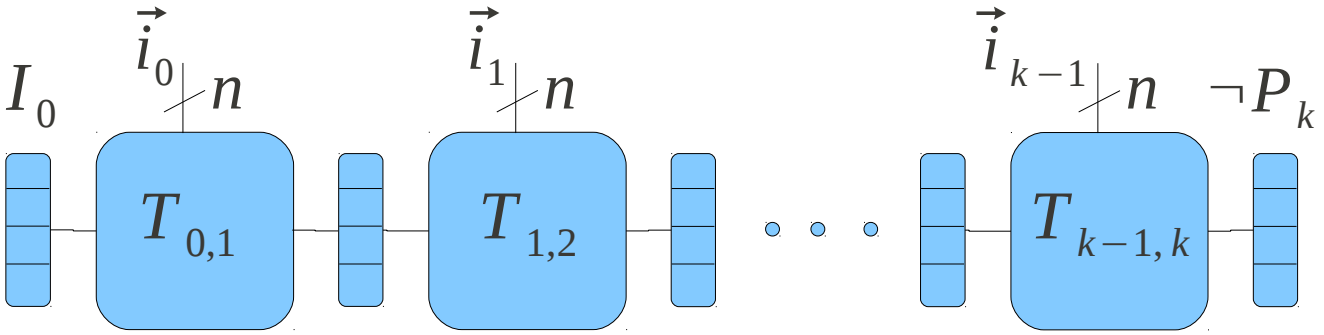
- BMC inputs:
  - Initial state I_0
  - Sequential circuit $T_{i,i+1}$
  - Property P_k



- Question: Can we reach $\neg P_k$ after k steps?

BMC (cont'd)

- Unrolling the circuit *k* times



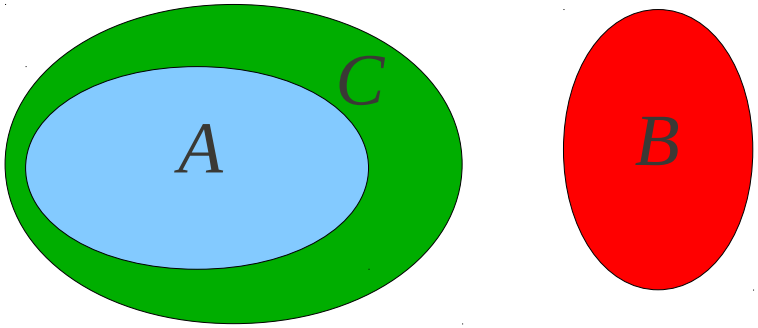
- Encode behaviour as a SAT problem

$$BMC_k = I_0 \wedge T_{0,1} \wedge \cdots \wedge T_{k-1,k} \wedge \neg P_k$$

- Satisfiable iff circuit has error trace of length *k*
- If no error trace is found, increment unroll depth

Craig Interpolation

- Craig interpolant theorem:
  - Let A and B be two clause sets with the property
    - $A \Rightarrow \neg B$ is valid
  - Then there exits a Craig interpolant C
    - C contains only *global variables*
    - $A \Rightarrow C$
    - $C \Rightarrow \neg B$
- Craig interpolant is an overapproximation:



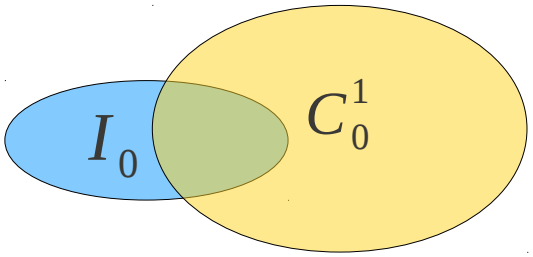
BMC + Craig Interpolation

- Craig interpolants can find a fixed point of reachable states [McMillan 03]

$$\underbrace{I_0 \wedge T_{0,1}}_{A} \wedge \underbrace{T_{1,2} \wedge \cdots \wedge T_{k-1,k} \wedge \neg P_k}_{B} \quad \text{unsatisfiable}$$

C_1^1 overapprox. of reachable states

- Apply fixed point check (FPC)
  - Check whether the C_1^1 contains new states $C_0^1 \Rightarrow I_0$
  - If valid the system is safe
  - If not valid inc. unroll depth



BMC + Craig Interpolation (cont'd)

- Inc. unroll depth

$$\underbrace{I_0 \wedge T_{0,1}}_{A} \wedge \underbrace{T_{1,2} \wedge \cdots \wedge T_{k-1,k} \wedge \neg P_k}_{B}$$

$$\Downarrow$$

$$C_1^1$$

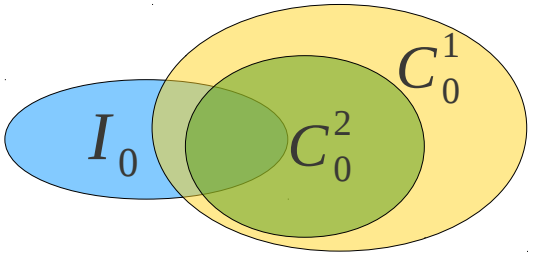
$$\underbrace{C_0^1 \wedge T_{0,1}}_{A} \wedge \underbrace{T_{1,2} \wedge \cdots \wedge T_{k-1,k} \wedge \neg P_k}_{B}$$

$$\Downarrow$$

$$C_1^2$$

- If unsat. compute next interpolant and FPC

$$C_0^2 \Rightarrow I_0 \vee C_0^1$$



- If satisfiable the counter example is maybe spurious
  - Perform a reset

$$\underbrace{I_0 \wedge T_{0,1}}_{A} \wedge \underbrace{T_{1,2} \wedge \cdots \wedge T_{k-1,k} \wedge T_{k,k+1} \wedge \neg P_{k+1}}_{B}$$

Accelerating BMC

- Incremental SAT-Solver [Een, Sörensson 03]
  - Reuse of learnt conflict clauses
  - Reuse of literal activities
- Preprocessing SAT-instances [Een, Biere 05]
  - Less clauses, less variables
  - Resolution, subsumption, blocked clause elim.

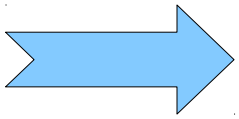
- **Problem:** How can we combine both?

Preprocessing in SAT

- CNF simplification:
  - Elimination of variables (resolution)
  - Literal elimination (self subsumption)
  - Clause deletion (subsumption, blocked clause elimination)
- Issues with incremental SAT solvers:
  - Blocked clauses may not stay blocked
  - New clauses containing previously eliminated variables may be added

Our Approach

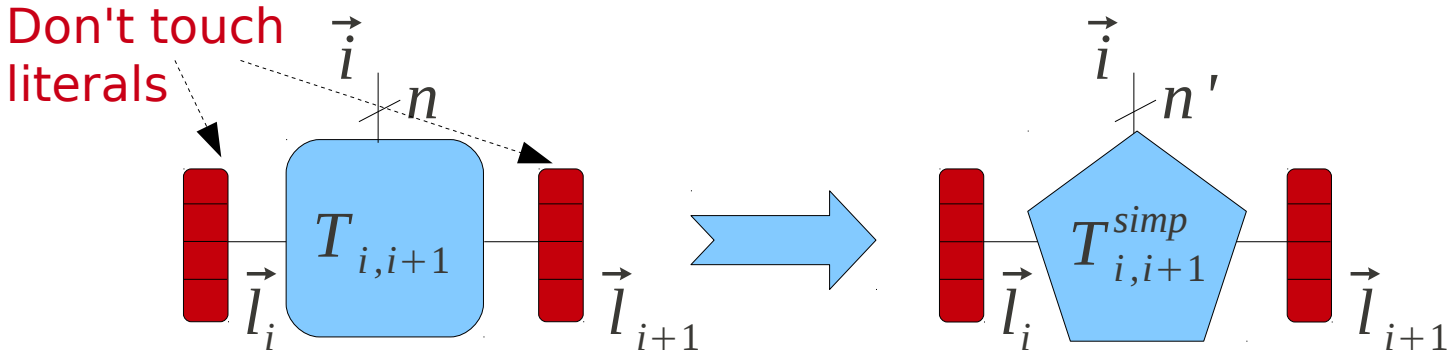
- **Idea:** Do not modify the “interface” of the circuit
- Preprocess the different BMC-parts
- Don't delete variables contained in future clauses
  - In BMC these are the latch variables
  - E.g. only literals that are not contained in future clauses are tested during blocked clause elim.



Doing this we can apply preprocessing to $T_{i,i+1}$ and can still use the simplified $T_{i,i+1}$ to create the correct BMC unrollings

Our Approach (cont'd)

- Preprocessor with don't touch literals



Our Approach (cont'd)

- Independent of the gen. of Craig interpolants

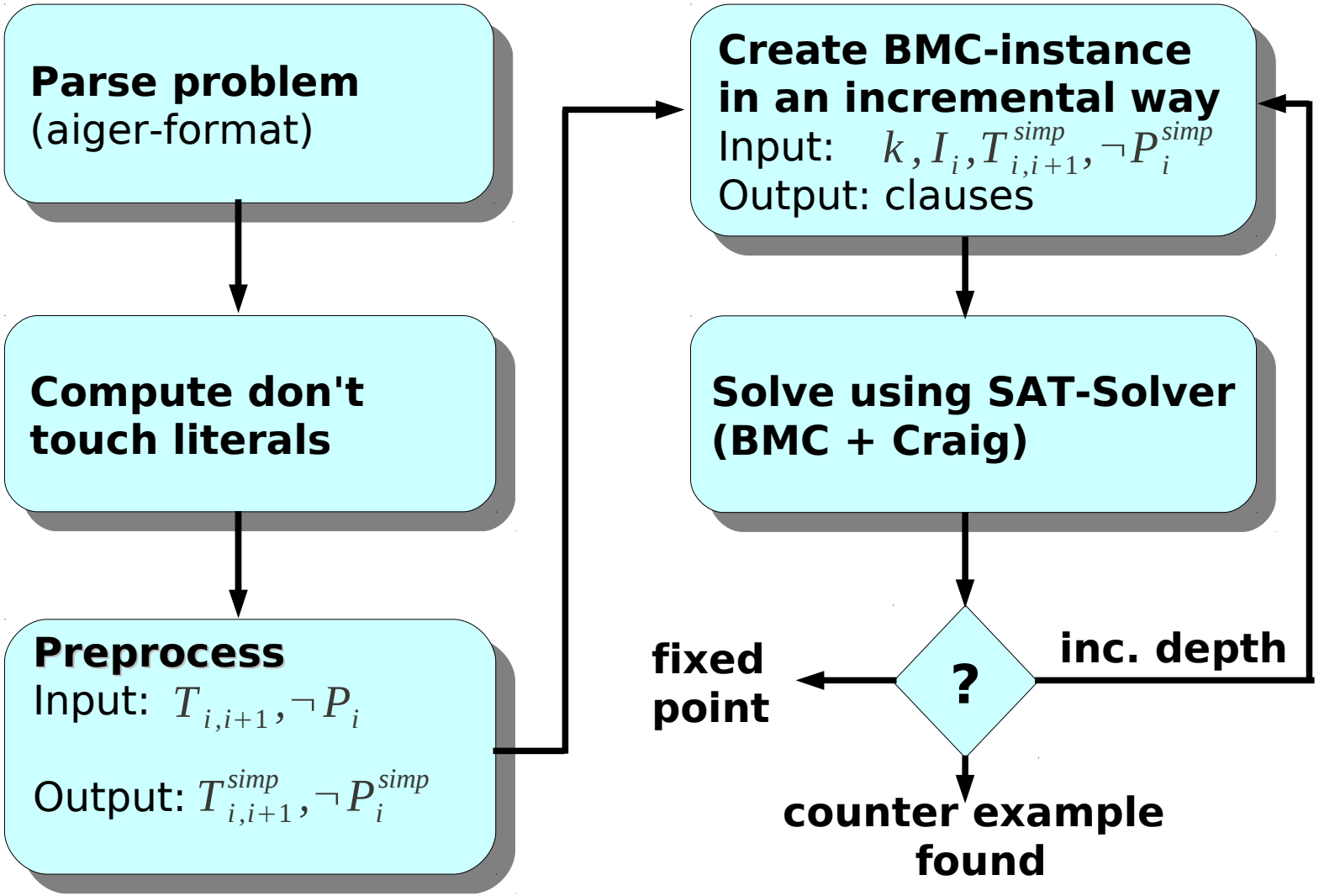
$$\underbrace{I_0 \wedge T_{0,1}^{simp}}_{A} \wedge \underbrace{T_{1,2}^{simp} \wedge \cdots \wedge T_{k-1,k}^{simp} \wedge \neg P_k}_{B}$$

- If unsat we compute *C* with: $A \Rightarrow C, C \Rightarrow \neg B$
- We know $T_{i,i+1} \Rightarrow T_{i,i+1}^{simp}$, and hence:
  - $I_0 \wedge T_{0,1} \Rightarrow I_0 \wedge T_{0,1}^{simp} \Rightarrow C$
  - $C \Rightarrow \neg\left(T_{1,2}^{simp} \wedge \cdots \wedge T_{k-1,k}^{simp} \wedge \neg P_k\right) \equiv$

    $C \Rightarrow \neg T_{1,2}^{simp} \vee \cdots \vee \neg T_{k-1,k}^{simp} \vee P_k \Rightarrow$

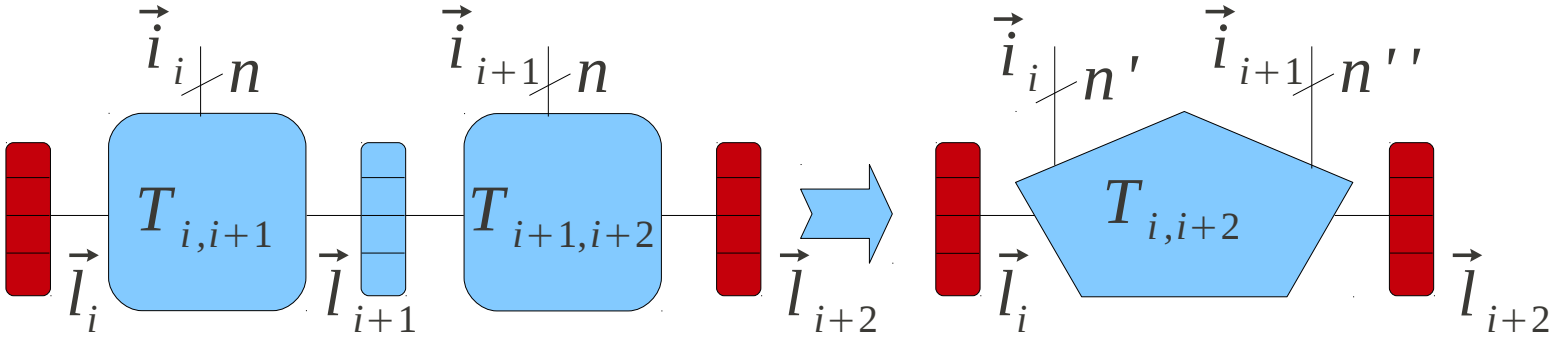
    $C \Rightarrow \neg T_{1,2} \vee \cdots \vee \neg T_{k-1,k} \vee P_k$

Workflow


Parse problem
(aiger-format)
Compute don't touch literals
Preprocess
Input: $T_{i,i+1}, \neg P_i$
Output: $T^{simp}_{i,i+1}, \neg P^{simp}_i$
Create BMC-instance in an incremental way
Input: $k, I_i, T^{simp}_{i,i+1}, \neg P^{simp}_i$
Output: clauses
Solve using SAT-Solver (BMC + Craig)
?
fixed point
inc. depth
counter example found

Advantages

- Only $T_{i,i+1}$ is preprocessed
- We can use an incremental SAT-solver
- Preprocessing does not affect the generation of Craig interpolants
  - Only resolution on “global variables” influences the gen. of interpolants (these are don't touch literals)
- Applicable to k-induction
- Preprocess more than one transition step



Experimental Results

- Our implementation:
  - Preprocessor taken from MiraXT
  - BMC tool based on SAT solver MiraXT
  - BMC + Craig is based on MiniSAT2
  - Total time is split between BMC and BMC + Craig
- Setup
  - 645 benchmarks taken from HWMCC'08
  - Quadcore Intel Q9450 processor @ 2.66GHz
  - 4GB of RAM
  - Timeout 900sec

Preprocessing Results

	Solver wo preprocessing	Solver w preprocessing
#clauses	8,723,774	3,915,462
#variables	5,462,710	1,710,189
time (sec)	9,345.07	4,540.71

- With don't touch literals the reduction of clauses/variables is still very good
- Average time was < 0.2s
  - Max. preprocessing time was only 5.8s
- Overall solving time was divided by 2

Experimental Results

- Comparison to the winners of the last HWMCC
  - TIP found most sat problems
  - ABC found most uns problems

	Our Solver	ABC	TIP
#uns solved	282	**314**	294
#sat solved	**253**	238	246
#total solved	535	**552**	540
total time (sec)	109,730.24	87,622.84	102,843.37

Experimental Results (cont'd)

Benchmark	S/U	#Vars.	#Cla.	Our Solver	ABC	TIP
intel048	-	261,275	685,929	TO	TO	TO
intel013	-	193,730	506,572	TO	TO	TO
intel039	sat	127,308	328,436	370.83	TO	TO
intel040	sat	125,386	322,616	379.48	TO	TO
intel041	sat	125,377	324,013	376.26	TO	TO
intel038	sat	122,600	317,149	371.68	TO	TO
intel042	sat	122,375	316,488	423.18	TO	TO
intel028	-	107,502	280,941	TO	TO	TO
intel043	sat	104,349	272,697	624.94	TO	TO
intel036	sat	98,327	262,244	590.42	TO	TO

- Our Solver (16/24), TIP (4/24), ABC (0/24)

Comparing Benchmark Families

Bench. Fam.	Best Solver
139*	Our Solver
ab*	ABC
bc57*	TIP
bj*	ABC
br*	Our Solver
cmu*	Our Solver
count*	Our Solver
cs*	Our Solver
dm*	Our Solver
eijk*	ABC
intel*	Our Solver
irst*	TIP

Bench. Fam.	Best Solver
ken*	Our Solver
mutex*	Our Solver
nec*	Our Solver
nus*	Our Solver
pc*	Our Solver
pdt*	ABC
prod*	Our Solver
ring*	TIP
short*	TIP
srg*	Our Solver
texas*	TIP
vis*	ABC

- Our Solver (14/24), TIP (5/24), ABC (5/24)

Conclusion

- Preprocessing with don't touch literals
  - Accelerates the verification process
  - Independent of the gen. of Craig interpolants
- Our tool is a first prototype
  - Optimizations are still possible
  - First results are promising

- To do:
  - Apply preprocess to more than one transition step
  - Test our approach with k-induction