

# Model Checking with Boolean Satisfiability

---

Joao Marques-Silva  
Electronics & Computer Science  
University of Southampton

University of Bristol, February 2007

# Motivation

- Remarkable improvements made to SAT solvers over the last decade
  - Clause learning; lazy data structures; adaptive branching heuristics; search restarts
- Very successful application of SAT in model checking
  - Bounded and unbounded model checking
- Existing (industry motivated) challenges
  - Ability to handle ever increasing systems
  - Ability to find deep counterexamples
  - Ability to prove difficult properties
- Lines of research
  - More efficient SAT solvers (?)
  - Better uses of SAT technology in SAT-based model checking

# Outline

- SAT overview
  - Organization of a modern SAT solver
  - Resolution, resolution proofs and interpolants
- SAT-based model checking
- Improvements to SAT-based model checking
- Results & conclusions

# What is SAT?

- Boolean satisfiability (SAT)
  - Given a propositional logic formula  $\varphi$ , decide whether there exist assignments to the propositional variables such that  $\varphi$  takes value true (or 1)
    - Propositional variables:  $x_1, x_2, \dots, x_n$
    - Standard logic connectives:  $\neg, \wedge, \vee, \leftrightarrow, \rightarrow$

$$(x_1 \vee x_2) \wedge (x_2 \rightarrow \neg x_3) = 1?$$

- If there are variable assignments that satisfy formula, then it is **satisfiable**; otherwise it is **unsatisfiable**
- Archetype NP-complete problem [Cook'71]
  - All known algorithms are exponential on the size of the representation, in the worst case
  - In practice:
    - Very effective algorithms developed over the last decade
      - Which can solve instances with millions of variables

# Applications of SAT

- Many applications and many successful applications
  - Artificial Intelligence
    - Planning; Knowledge compilation; ...
  - Electronic Design Automation
    - Hardware model checking
      - Arguably the most successful application of SAT
    - Equivalence checking; Test-pattern generation; Fault diagnosis; ...
  - Software Engineering
    - Software model checking; Software testing; ...
  - Computational Biology
    - Haplotype inference; Pedigree consistency; ...
  - Theorem proving
  - Answer set programming
  - Description logics
  - ...

# Extensions of SAT

- Success of SAT motivated work on extensions of SAT
  - Pseudo-Boolean Optimization (PBO) / 0-1 ILP
    - Conjunctions of linear inequalities over Boolean variables
  - Quantified Boolean Formulas (QBF)
    - Propositional logic with quantifiers
  - Satisfiability Modulo Theories
    - Decidable theories (ILA, RLA, ...)
      - ILA: conjunction, disjunction and negation of linear inequalities over the integers
      - ...

# CNF formulas

- Conjunctive normal form (CNF):
  - Standard representation for SAT
  - CNF formula  $\varphi$  is a conjunction of clauses
  - Clause is a disjunction of literals
  - Literal is a variable or its complement

$$\varphi = (a \vee b) \wedge (\neg a \vee c) \wedge (c \vee \neg d \vee \neg e) \wedge (\neg d \vee \neg a)$$

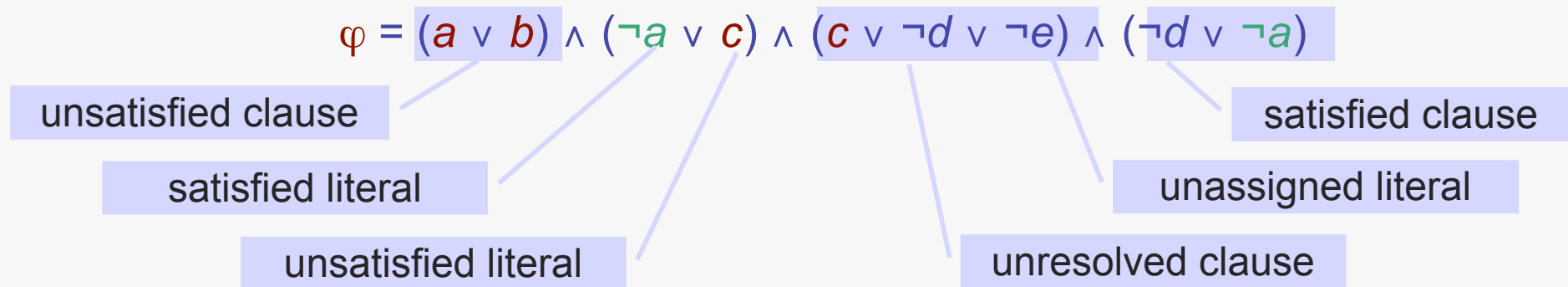
$$\varphi = (a \vee b) (\neg a \vee c) (c \vee \neg d \vee \neg e) (\neg d \vee \neg a)$$

- Can map propositional formulas into CNF in linear time
  - Addition of a linear number of auxiliary variables

[Tseitin'68; Plaisted&Greenbaum'86]

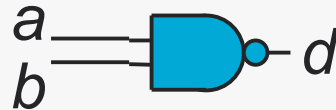
# CNF formulas

- Given a partial assignment to the variables:
  - A literal is **satisfied** if its value is 1; it is **unsatisfied** if its value is 0; otherwise it is unassigned
  - A clause is **satisfied** if at least one of its literals is satisfied; it is **unsatisfied** if all of its literals are unsatisfied; otherwise it is unresolved
  - A formula is **unsatisfied** if at least one clause is unsatisfied; it is **satisfied** if all clauses are satisfied; otherwise it is unresolved



# Representing gates in CNF

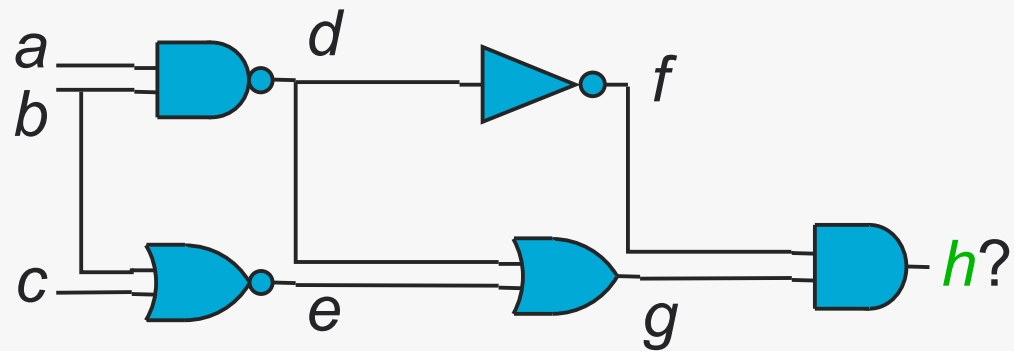
[Tseitin'68; Plaisted&Greenbaum'86]



$$\begin{aligned}\varphi_d &= [d = \neg(a b)] \\ &= \neg[d \oplus \neg(a b)] \\ &= \neg[\neg(a b)\neg d + a b d] \\ &= \neg[\neg a \neg d + \neg b \neg d + a b d] \\ &= (a + d)(b + d)(\neg a + \neg b + \neg d)\end{aligned}$$

# Representing circuits in CNF

[Tseitin'68; Plaisted&Greenbaum'86]



$$\varphi = h [d = \neg(ab)] [e = \neg(b+c)] [f = \neg d] [g = d+e] [h = fg]$$

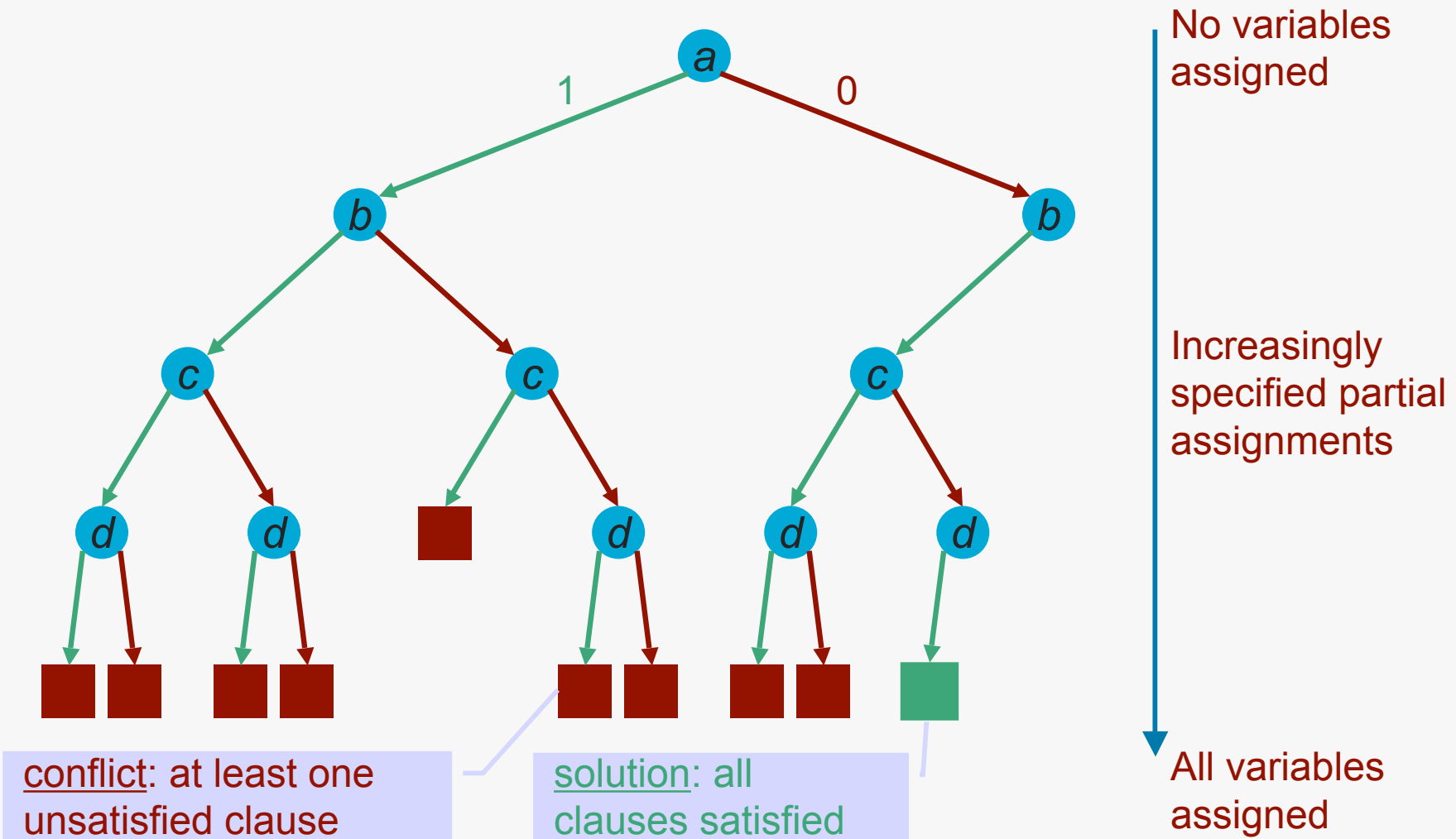
$\varphi$  is the **characteristic function** for circuit with output  $h$

# Algorithms for SAT

- Incomplete Algorithms (Cannot prove unsatisfiability)
  - Local search (hill climbing)
  - Lagrangian multipliers
  - Genetic algorithms
  - Simulated annealing
  - Tabu search
  - ...
- Complete Algorithms (Can prove unsatisfiability)
  - Backtrack search (DPLL)
  - Resolution
  - Stalmarck's method
  - Recursive learning
  - Binary decision diagrams (BDDs)
  - ...
- The utilization of SAT in model checking requires ability to prove unsatisfiability
  - Most SAT algorithms used in model checking are based on backtrack search

# Plain backtrack search

- Given a CNF formula  $\varphi$ , i.e. a conjunction of clauses, implicitly enumerate all partial assignments to the variables



# Unit propagation

[Davis&Putnam'60]

- Unit clause:
  - A clause  $\omega$  is unit iff all literals but one are assigned value 0 and one literal is unassigned
    - With  $a = 0$  and  $b = 1$ ,  $\omega = (a \vee \neg b \vee c)$  is unit
- Unit clause rule:
  - If a clause  $\omega$  is unit, then unassigned literal must be assigned value 1
    - With  $a = 0$  and  $b = 1$ ,  $\omega = (a \vee \neg b \vee c)$  is unit
    - Literal  $c$  must be assigned value 1 for  $\omega$  to be satisfied
      - With  $c = 1$ ,  $\omega = (a \vee \neg b \vee c)$  becomes satisfied
- Unit propagation:
  - Iterative application of the unit clause rule
    - Imply variable assignments until no more unit clauses, or unsatisfied clause is identified

# The DPLL algorithm

[Davis et al.'62]

- Backtrack search
  - Implicit enumeration of all partial assignments
- Unit propagation
  - Iterated application of unit clause rule
- Variable selection heuristic
  - Policy for selecting the variable to branch on and the value to assign the variable
- DPLL seldom used in practical applications until the mid 90s !

# Modern SAT algorithms

- Follow the organization of the DPLL algorithm [Davis et al.'62]
  - Backtrack search with unit propagation
- Several key techniques are used:
  - Clause learning [Marques-Silva&Sakallah'96]
    - Infer new clauses from causes of conflicts
      - Allows implementing non-chronological backtracking
  - Exploiting structure of conflicts [Marques-Silva&Sakallah'96]
    - Unique Implication Points (UIPs)
      - Dominators in graph of implied assignments
  - Optimised data structures [Moskewicz et al.'01]
    - Lazy evaluation of clause state
  - Adaptive branching heuristics [Moskewicz et al.'01]
    - Variable branching metrics are affected by number of conflicts
    - Aging mechanisms for focusing on most recent conflicts
  - Search restarts [Gomes,Selman&Kautz'98]
    - Opportunistically restart backtrack search

# Clause learning

- During backtrack search, for each conflict learn clause that explains and prevents repetition of same conflict

$$\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f) \dots$$

Assume (decisions)  $c = 0$  and  $f = 0$

Assign  $a = 0$  and imply assignments

A conflict is reached:  $(\neg d \vee \neg e \vee f)$  is unsatisfied

$$(a = 0) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$$

$$(\varphi = 1) \Rightarrow (a = 1) \vee (c = 1) \vee (f = 1)$$

$\therefore$  learn new clause:  $(a \vee c \vee f)$

# Non-chronological backtracking

- During backtrack search, in the presence of conflicts, backtrack to one of the causes of the conflict

$$\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f) \\ (a \vee c \vee f)(\neg a \vee g)(\neg g \vee b)(\neg h \vee j)(\neg i \vee k) \dots$$

Assume (decisions)  $c = 0$ ,  $f = 0$ ,  $h = 0$  and  $i = 0$

Assignment  $a = 0$  caused conflict  $\Rightarrow$  learned clause  $(a \vee c \vee f)$   
 $(a \vee c \vee f)$  implies  $a = 1$

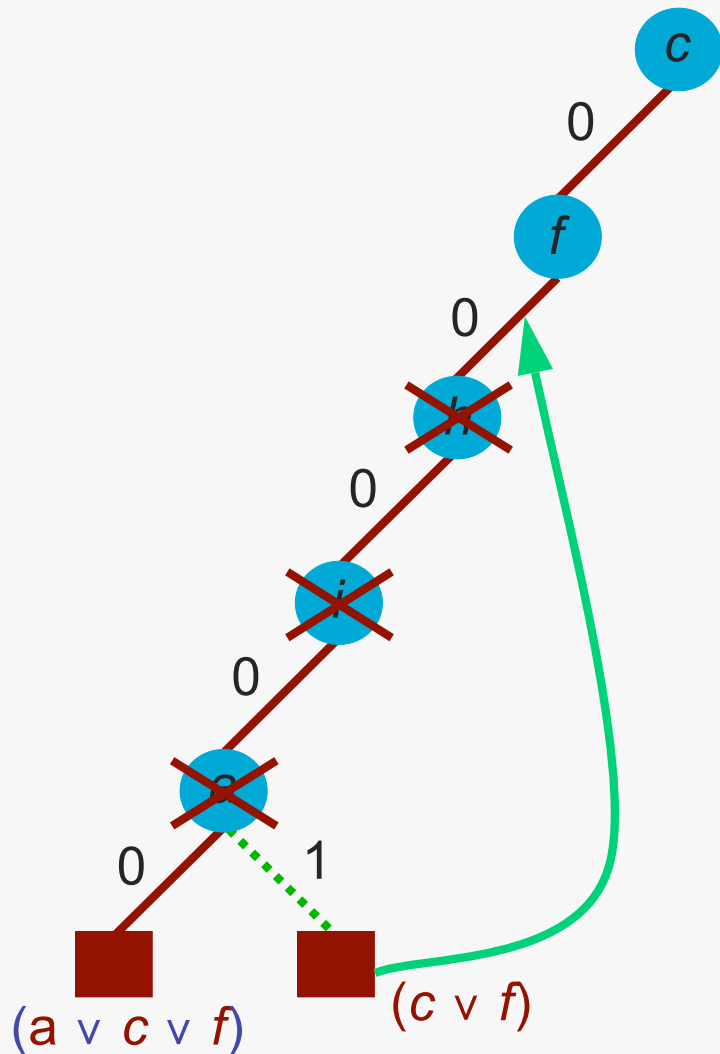
A conflict is again reached:  $(\neg d \vee \neg e \vee f)$  is unsatisfied

$$(c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$$

$$(\varphi = 1) \Rightarrow (c = 1) \vee (f = 1)$$

**$\therefore$  learn new clause:  $(c \vee f)$**

# Non-chronological backtracking



Learned clause  $(c \vee f)$

Need to backtrack, given  $(c \vee f)$

Backtrack to most **recent**  
decision:  $f = 0$

$\therefore$  Clauses learned:  
 $(a \vee c \vee f)$  and  $(c \vee f)$

In practice, learned clauses can allow backtracking over a significant percentage of the decision variables

# Evolution of SAT solvers

- Remarkable improvements over the last decade

Instance	Posit' 94	Grasp' 96	Chaff'01	Siege'04
ssa2670-136	28.53	0.36	0.02	0.01
bf1355-638	772.45	0.04	0.02	0.01
design_1	>7200	65.35	1.27	0.29
design_3	>7200	9.13	0.52	0.41
f_ind	>7200	4663.89	17.91	6.52
splitter_42	>7200	>7200	28.81	4.46
c6288	>7200	>7200	>7200	2847.46
pipe_64_32	>7200	>7200	>7200	>7200

# Resolution

[Robinson'65]

- Refutation-complete procedure for first order logic
- In propositional logic:
  - Technique for deriving new clauses
    - Example:  $\omega_1 = (\neg a \vee b \vee c)$ ,  $\omega_2 = (a \vee b \vee d)$
    - Resolution:

$$\text{res}(\omega_1, \omega_2, a) = (b \vee c \vee d)$$

[Davis&Putnam'60]

- Forms the basis of a complete procedure for satisfiability
- Impractical for real-world formulas
- Application of restricted forms has been successful
  - E.g., restricted resolution
    - $\text{res}((\neg a \vee \alpha), (a \vee \alpha), a) = (\alpha)$   
 $\alpha$  is a disjunction of literals

# Resolution refutations

- Clause learning can be viewed as the inference of a clause by a sequence of resolution steps

$$\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f) \dots$$

- $a = 0$  yields conflict; can learn  $(a \vee c \vee f)$
- By applying resolution:

$$\varphi = (a \vee b)(\neg b \vee c \vee d)(\neg b \vee e)(\neg d \vee \neg e \vee f) \dots$$

resolution

$$(a \vee c \vee d)$$

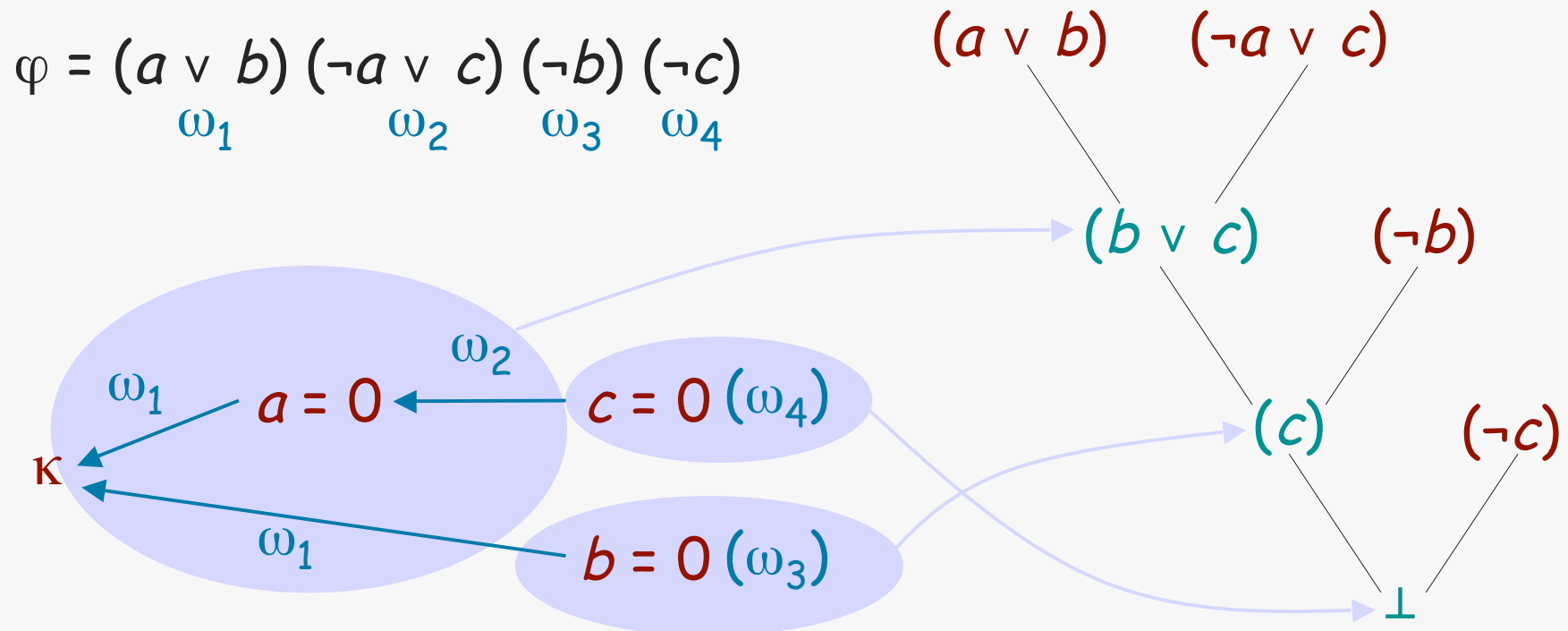
$$(a \vee e)$$

$$(a \vee c \vee \neg e \vee f)$$

$$(a \vee c \vee f)$$

# Deriving resolution refutations

- For unsatisfiable formulas: [Zhang&Malik'03]
  - Learned clauses capture a resolution refutation from a subset of the original clauses
  - SAT solvers can be instructed to recreate resolution refutation for unsatisfiable formula



# Interpolants

[Craig'57]

- Given two subsets of clauses  $A$  and  $B$ , assume  $A \wedge B$  is unsatisfiable. Then, there exists an interpolant  $A'$  for the pair  $(A, B)$  with the following properties:
  - $A$  implies  $A'$
  - $A' \wedge B$  is unsatisfiable
  - $A'$  refers only to the common variables of  $A$  and  $B$
  - Example:
    - $A = p \wedge q, B = \neg q \wedge r$
    - $A' = q$

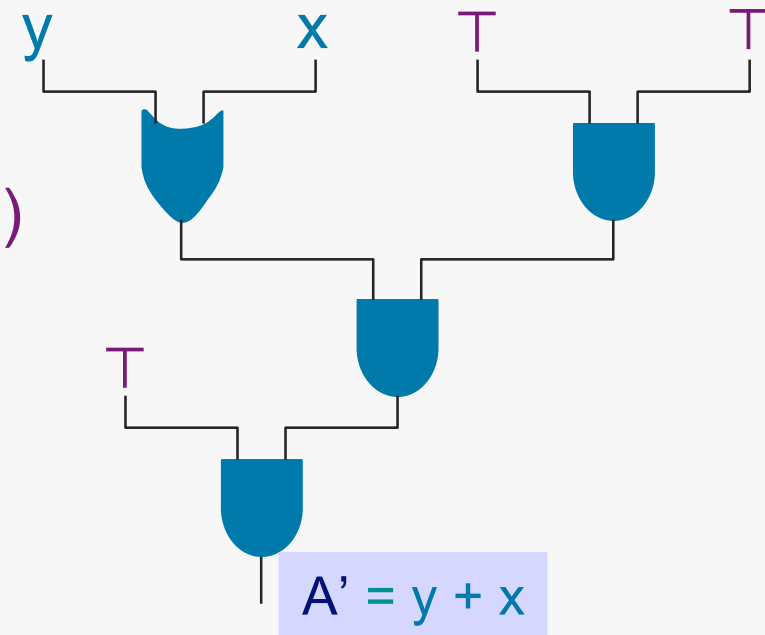
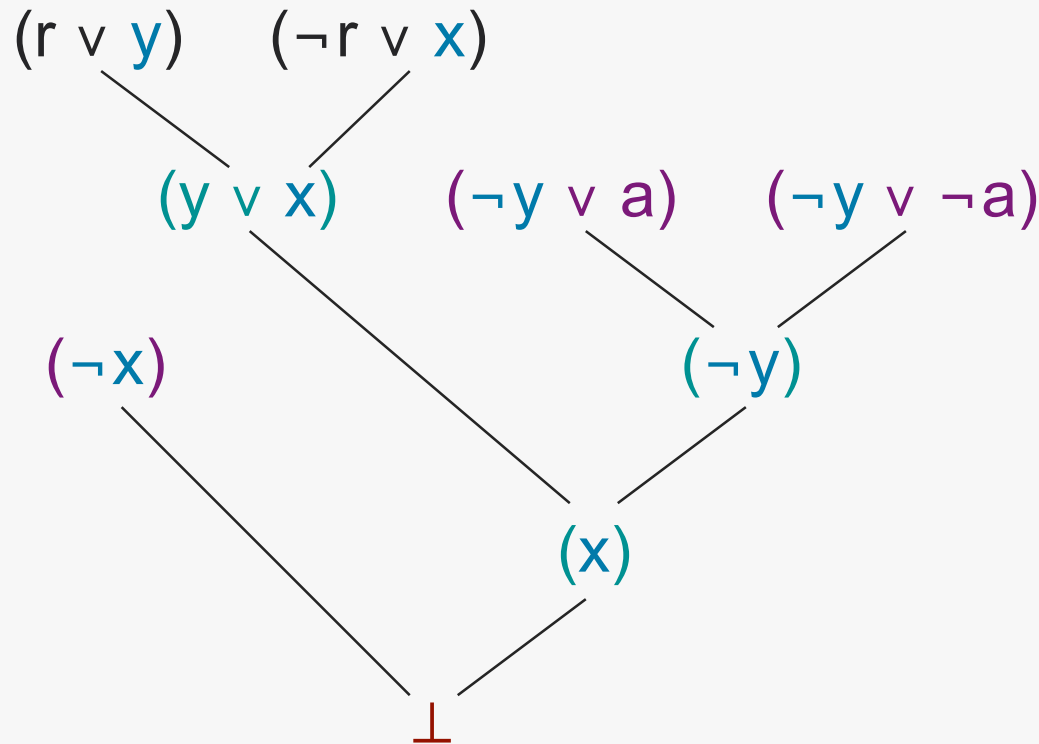
- Size of interpolants: [Pudlak'97]
  - Given a resolution refutation of  $A \wedge B$ , can compute interpolant for the pair  $(A, B)$  in linear time on the size of the resolution refutation
    - SAT solvers can be instructed to output resolution refutation !

- Computing interpolants:
  - Different algorithms can be used [McMillan'03]
    - Pudlak'97, McMillan'03

# Computing interpolants

$$A = (r \vee y)(\neg r \vee x)$$

$$B = (\neg y \vee a)(\neg y \vee \neg a)(\neg x)$$



$A$  implies  $A'$ ;  $A' \wedge B$  is unsatisfiable  
 $A'$  with variables common to  $A$  and  $B$

- Interpolant is a Boolean circuit that follows structure of resolution refutation
  - Can map circuit into CNF in linear time and space

[Tseitin'68; Plaisted&Greenbaum'86]

# Outline

- SAT overview
- SAT-based model checking
  - SAT-based bounded model checking (BMC)
  - Interpolant-based unbounded model checking (UMC)
- Improvements to SAT-based model checking
- Results & conclusions

# Bounded model checking

[Biere et al.'99]

- Verification of **safety** properties: **F f**

$$\Phi^k = I_0(Y_0) \wedge \bigwedge_{i=0}^{k-1} T(Y_i, Y_{i+1}) \wedge \left( \bigvee_{i=r}^k f(Y_i) \right)$$

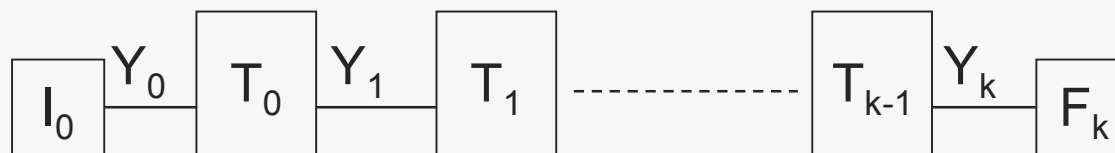
- **Characteristic functions** for representing initial states and transition relation, respectively **I<sub>0</sub>** and **T**

– Resulting Boolean formula:  $\Phi^k = I_0 \wedge U_k \wedge F_k$

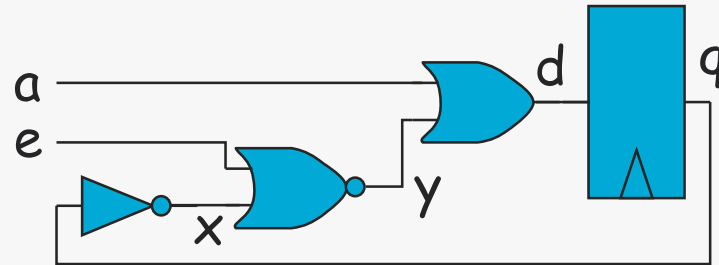
- Where:

$$U_k = \bigwedge_{j=0}^{k-1} T_j \quad T_j = T(Y_j, Y_{j+1}) \quad F_k = \left( \bigvee_{i=r}^k f_i \right) \quad f_i = f(Y_i)$$

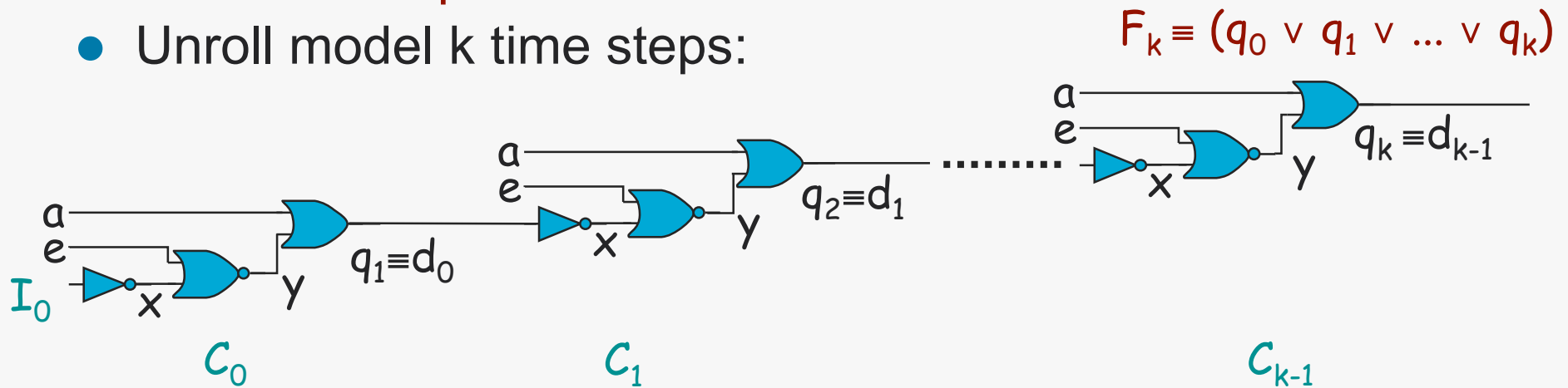
– Interpretation:



# An example



- Property:  $G \neg q$  ?
- Evaluate:  $F q$
- Unroll model k time steps:



- Check satisfiability of CNF formula for  $I_0 \wedge U_k \wedge F_k$

# Bounded model checking

- A possible BMC algorithm:
  - Given some initial  $k$
  - While  $k \leq$  user-specified time-bound UB BMC loop
    - Generate CNF formula  $\varphi$  for  $I_0 \wedge U_k \wedge F_k$
    - Invoke SAT solver on  $\varphi$
    - If formula  $\varphi$  is satisfiable, then a counterexample within  $k$  time steps has been found
      - Return counterexample
    - Otherwise, increase  $k$
- The BMC algorithm is **incomplete**
  - But complete if completeness threshold is known

# Towards completeness

- Unbounded model checking

- Utilization of induction

[Sheeran et al.'00]

- Standard BMC loop

- Stop BMC loop for some  $i$ , if cannot have loop-free path of size  $i$  that can be reached from  $I_0$  or if cannot have loop-free path of size  $i$  that can reach  $F_k$
      - Maximum unfolding bounded by largest loop-free path

- ...

[Chauhan et al.'02; Gupta et al.'03]

- Utilization of interpolants

[McMillan'03]

- BMC and Craig interpolants allow SAT-based computation of abstractions of reachable states

- Avoid computing exact sets of reachable states
      - One of the most promising approaches in practice
        - Maximum unfolding bounded by largest shortest path between any two states

# Abstraction of reachable states

- For each iteration of BMC loop, call to SAT solver returns unsat unless counterexample is found
  - Analysis of resolution refutation yields abstractions of reachable states

$$\Phi = I_0 \wedge T_0 \wedge T_1 \wedge \dots \wedge T_{k-1} \wedge F_k = A \wedge B$$

$$A = I_0 \wedge T_0$$

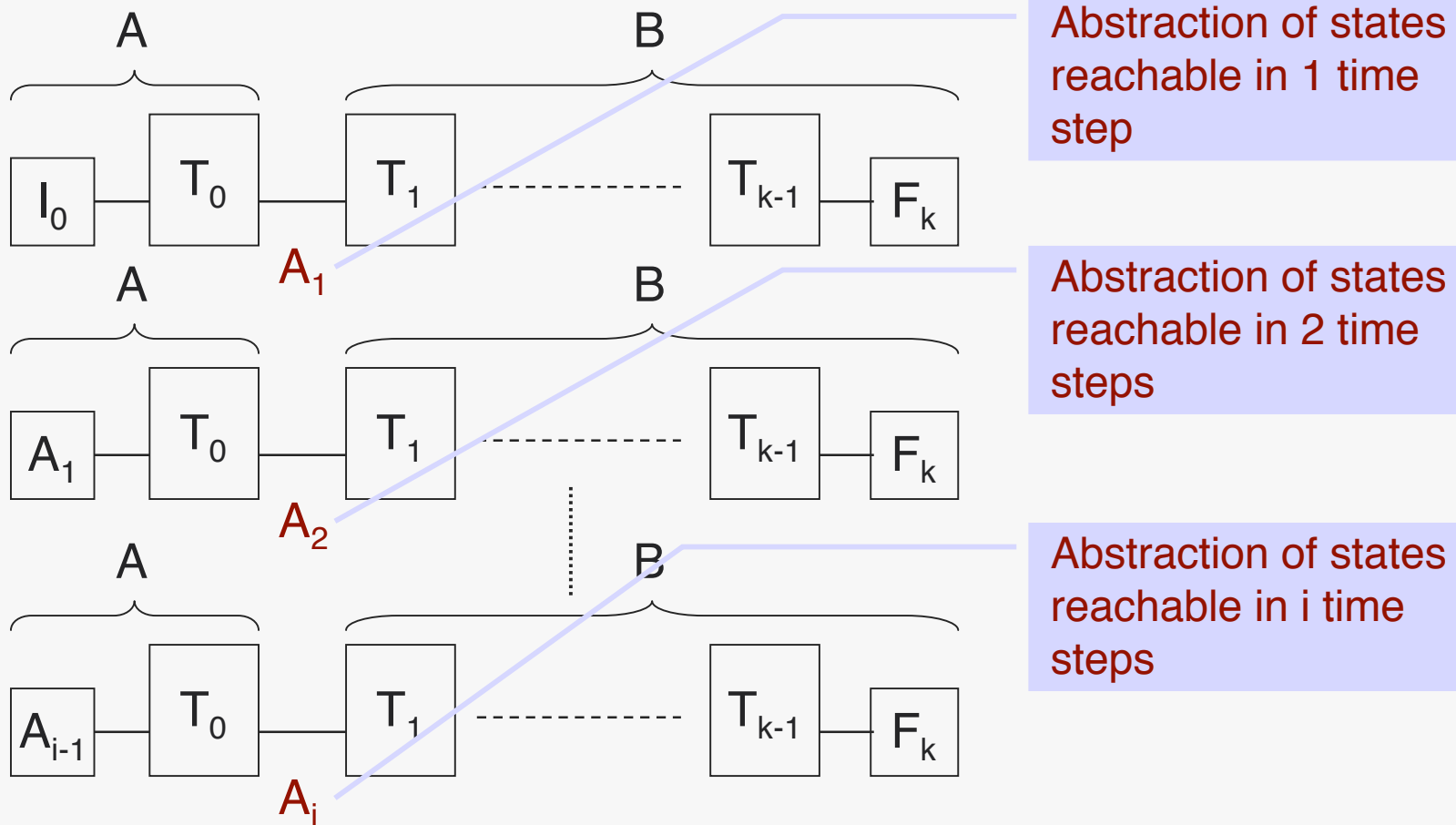
$$B = T_1 \wedge \dots \wedge T_{k-1} \wedge F_k$$

- Given A and B, and a resolution refutation for  $A \wedge B$ , compute Craig interpolant A':

- $A = I_0 \wedge T_0$  **implies** A'
- A'  $\wedge$  B is unsatisfiable
- A' solely represented with **state** variables
  - If A holds, then A' holds
    - $A_1 = A'$  represents abstraction of states reachable from  $I_0$  in 1 time step !

# Fixpoint of reachable states

- Can **iterate** computation of interpolants:



If  $A_i \rightarrow I_0 \vee A_1 \vee A_2 \vee \dots \vee A_{i-1}$ , then a **fixpoint** is reached; all reachable states identified !

# UMC algorithm

$k = 0$

repeat

if from  $I_0$  can satisfy  $F_k$  within  $k$  steps

return **reachable**

$R = I_0$

let  $A = I_0 \wedge T_0$ , and  $B = T_1 \wedge T_2 \wedge \dots \wedge T_{k-1} \wedge F_k$

while  $A \wedge B = \text{false}$

$P = \text{unsat\_proof}(A \wedge B)$

$A' = \text{interpolant}(P, A, B)$

if  $A' \rightarrow R$ , return **unreachable**

$R = A' \vee R$

$A = A' \wedge T_0$

end while

increase  $k$

end repeat

**BMC loop**

If  $F_k$  is satisfied from  $I_0$ , then we have a **counterexample** !

**Fixpoint**

If a fixpoint of the reachable states is identified, then **no** reachable state can satisfy property !

Calls to SAT solver

If  $A \wedge B$  is sat, may have abstracted too much; must unfold more time steps

Maximum value of  $k$  is bounded by **largest shortest path** between any two states

# Outline

- SAT overview
- SAT-based model checking
- Improvements to SAT-based model checking
- Results & conclusions

# Rescheduling the BMC loop

[Marques-Silva'05]

$k = 0$

repeat

if from  $I_0$  can satisfy  $F_k$  within  $k$  steps  
return **reachable**

$R = I_0$

let  $A = I_0 \wedge T_0$ , and  $B = T_1 \wedge T_2 \wedge \dots \wedge T_{k-1} \wedge F_k$

while  $A \wedge B = \text{false}$

**Fixpoint**

$P = \text{unsat\_proof}(A \wedge B)$

$A' = \text{interpolant}(P, A, B)$

if  $A' \rightarrow R$ , return **unreachable**

$R = A' \vee R$

$A = A' \wedge T_0$

end while

increase  $k$

end repeat

**BMC loop**

Number of iterations can be used to restrict when to check again the BMC condition !

# Rescheduling the BMC loop

```
while A  $\wedge$  B = false Fixpoint  
  P = unsat_proof(A  $\wedge$  B)  
  A' = interpolant(P, A, B)  
  if A'  $\rightarrow$  R, return unreachable  
  R = A'  $\wedge$  R  
  A = A'  $\wedge$  T0  
end while
```

Fixpoint checking with  $i+1$  iterations (last iteration is sat):

$$I_0 \longrightarrow A_1 \longrightarrow A_2 \longrightarrow \dots \longrightarrow A_{i+1}$$

Checked **all** states reachable in up to  $k+i$  states, with an unfolding of size  $k$ ; no counterexample was found

$\therefore$  Need to check **BMC** condition **only** when unfolding of FSM exceeds  **$k+i$**  time steps

In general useful if counterexample exists

# Results on rescheduling

- Evaluated rescheduling on different benchmarks
  - Specifically designed and industrial examples
- Evaluated both the plain UMC algorithm and rescheduling

Instance	No-reschedule	Reschedule BMC
4-bit counter	0.31	0.09
5-bit counter	3.86	0.84
6-bit counter	21.36	10.41
7-bit counter	1780.68	175.69
I12	255.77	272.47
I11	75.28	81.89
I31	83.51	90.08
I32	19.66	14.89
I33	17.44	13.09
I21	24.93	26.48
Total time	2282.8	685.9

# Conclusions

- SAT technology has improved dramatically over the last decade
  - Key techniques:
    - Clause learning, optimized data structures, adaptive branching heuristics, search restarts
- SAT has been applied to model checking with success
  - Bounded and unbounded model checking
- Optimisations to the use of interpolants